Dublin City University

Electronic and Computer Engineering

Streaming Audio Server with Listener-Tracking Embedded Clients

Author

Michael Lenehan michael.lenehan4@mail.dcu.ie

Supervisor

Martin Collier martin.collier@mail.dcu.ie

08/04/2019

Contents

List of Tables

List of Figures

1 Acknowledgements

I would like to thank my project supervisor, Dr. Martin Collier. He provided guidance and assistance with all aspects of this project, and his knowledge on the topic was invaluable. I would also like to thank Dr. Gabriel-Miro Muntean, who acted as the second assessor for this project for his insight into testing practices, which gave greater clarity to the overall project results.

2 Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf and IEEE referencing guidelines found at https://loop.dcu.ie/mod/url/view.php?id=448779.

Signed: _____ Date: 08/04/2019

Michael Lenehan

3 Abstract

With the widespread adoption of audio streaming services such as Spotify, Tidal, and Google Play Music, there is a need for a high quality audio streaming option for locally stored audio. Open-source audio server softwares can provide this functionality, although they are intended for use in "headless" audio setups, controlled over the network, outputting their audio locally. This project aims to implement an audio server solution which can stream music over the network to a number of client devices, at high quality, with minimal audio artefacts. Audio playback across the client devices must be synchronised in order to provide a satisfactory user experience. Open-source audio streaming software "Snapcast" is used in order to implement multi-output synchronised audio playback across a number of network connected client devices. In addition to synchronised audio playback, listener tracking will be implemented in such a way as to allow a user traverse a space, such as their home, with the audio level altered as the listener approaches or distances themselves from the client device. Finally, the system volume control and listener tracking will be combined in a cross-platform application, built using the "Flutter" framework.

4 Introduction

There are many options for open-source audio streaming available to users. A number of configurations of the available open-source audio streaming hardware and software allow for end-users to play locally stored audio on a so-called "Headless" system, whereby the server software is controlled remotely by the user.

In recent years, a movement away from locally stored audio solutions has taken place [4]. Subscription based audio streaming services such as Spotify exist to allow users access audio which they have no access to physical copies of. While this trend exists, an optimal server software solution would serve both locally stored, and streaming audio options to listeners. Offering both of these options allows for the user to have the choice of configuring a database of their own audio files, or streaming from a preconfigured streaming service.

This project explores the idea of implementing a streaming server which allows users access a stored collection of audio, from any connected device, and to stream this audio to the nearest available client device. A solution must offer an accessible user experience, and importantly provide good quality playback, achieved through the combination of chosen software and hardware.

Listener tracking implementations must be investigated. These implementations tend to include Wi-Fi or Bluetooth tracking through either signal strength, or Round Trip Time algorithms. More complex examples exist of implementations using multiple antenna arrays to give directionality and distance, however, these implementations tend to be cost prohibitive. The advantages and disadvantages of these implementations will be discussed, with an option chosen on its merits and availability.

Following the investigation into the available solutions, a model must be proposed with specifics given for the audio streaming, listener tracking, and Android application implementations. Testing procedures of the audio server software and listener tracking must be given, with all results recorded and analysed.

5 Background Literature Survey

5.1 Hardware Requirements

Embedded Linux Devices

There are a number of Linux based embedded systems which may be configured to act as an audio streaming server with the appropriate software. Commonly used systems include the Raspberry Pi 3 Model B+, BeagleBone Black, and ASUS Tinker Board. There are differences between these development platforms which allow them greater or less suitability for the purposes of this project.

The BeagleBone Black (BBB) is a low cost platform, with compatibility for many Linux distributions. The device has on board flash memory, Ethernet and HDMI outputs. There is also on board I^2S support, allowing for hardware Digital to Analog Converters (DACs) to be connected. The BBB has 512MB of DDR3 RAM, and a 1GHz ARM processor on board [1].

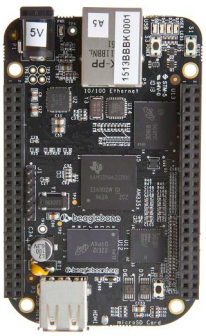

Figure 5.1: BeagleBone Black [1]

The ASUS Tinker Board is a small form-factor Single Board Computer (SBC). The computer has Gigabit Ethernet, HDMI output, multiple IO, including 40 GPIO pins and 4 USB ports. The 1.8GHz ARM based CPU provides high performance when coupled with the 600MHz GPU and 2GB of dual-channel DDR3 RAM. This SBC also supports the I^2S audio protocol [5].

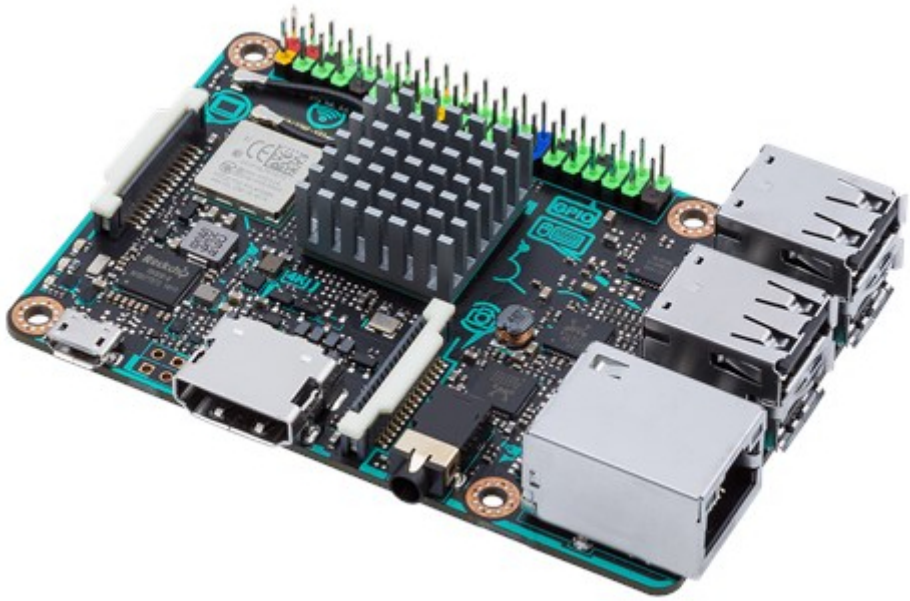

Figure 5.2: Asus Tinker Board
[5]

The Raspberry Pi 3 Model B+ is one of the most commonly used embedded Linux development platforms. The device has a 1.4GHz ARM processor, 1GB of DDR2 RAM, Gigabit Ethernet, Bluetooth Low Energy, and multiple IO ports. Again, this board supports the I^2S protocol, with outputs on its GPIO [2].

Figure 5.3: Raspberry Pi 3 Model B+ [2]

Each of the aforementioned options offers different levels of performance at different price points. The BeagleBone Black is both the cheapest and least powerful option. The ASUS

Tinker Board is the most powerful and most expensive option, while the Raspberry Pi offers comparatively high performance at a mid price. The benefits and costs of these Single Board Computers must be compared in order to choose that which is most appropriate for the application of serving and streaming audio.

For the purposes of this project, the Raspberry Pi Model 3 B+ was chosen. This platform has a large user base, and an active online community, allowing for a more user friendly experience, especially with regards to aspects such as initial setup and software installation. The Raspberry Pi also has a large number of available add-on boards, commonly referred to as "Hats" or "Bonnets". These boards may be used to give additional functionality to the Raspberry Pi, with boards available for purposes such as adding display capabilities, numerous sensors, and, as is applicable to this project, audio DAC and amplifiers.

Audio DAC

As previously mentioned, there are a number of amplifier and DAC add-on options for the Raspberry Pi 3 Model B+. Some of the most popular options available are from HiFiBerry, including their HiFiBerry Dac+ Pro. This board offers RCA output, with dual-domain low-jitter clocks [6].

The option chosen was the Adafruit I2S Audio Bonnet for Raspberry Pi. This DAC also utilises the I^2S audio protocol, through the UDA1334A stereo DAC. The board outputs audio via a standard 3.5mm audio jack, with the option of soldering RCA jacks to the PCB.

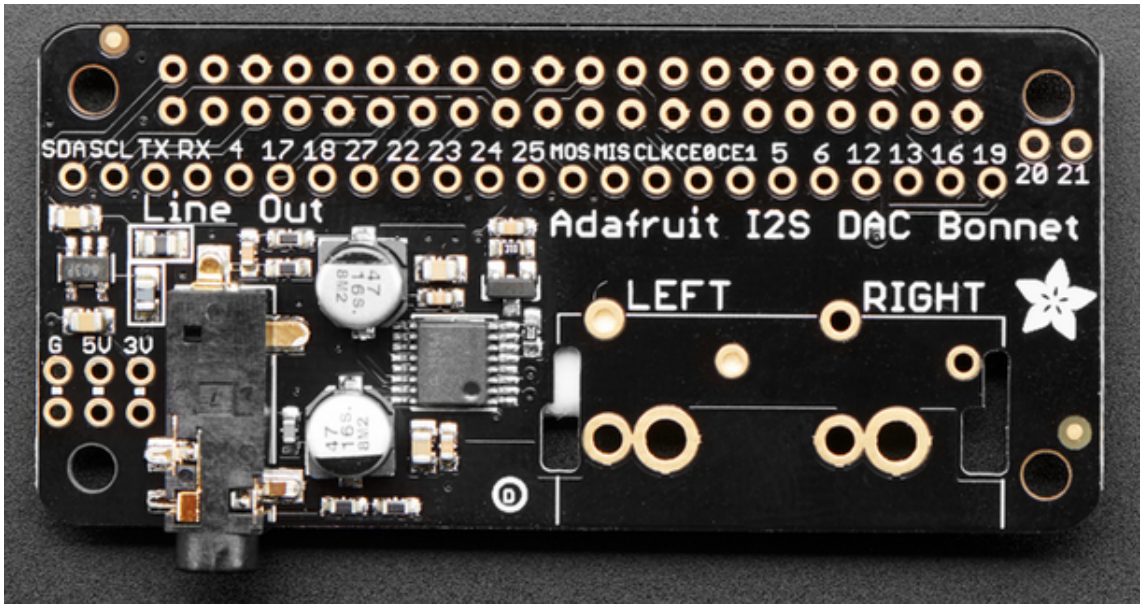

Figure 5.4: Adafruit I2S Audio Bonnet [3]

5.2 Open Source Software

A number of software solutions exist for streaming audio from low powered hardware. Options such as MPD - the Music Player Daemon -, Mopidy, and Volumio, allow for users to play music on the system. These options are typically used to implement headless audio

player setups, with the user sending messages over the network to control the player.

While these options provide much of the basic required functionality, they are not a suitable solution for the project. The required functionality from the software will be to stream media from the server system to the client system. This functionality exists in these open-source software options, but requires modifications to be made to configuration files in order to be implemented.

Audio software PulseAudio may also be utilised as it is often found on Linux based systems. PulseAudio is a sound server which routes audio from the running application to the selected output device. On Linux systems, this is used to send audio output to the system speakers, or connected USB devices. However, the functionality exists to pass the output audio over the network to a specified address [7].

MPD

MPD is a server-side audio application, available on Debian based Linux platforms, such as Raspbian, via the standard "apt" package manager repositories. Via the available configuration files, parameters such as the music directory, audio output device, audio encoder and decoder plugins, and audio format settings may be set. MPD allows for playback of a number of audio formats, including WAV, FLAC, and MP3, and supports both FIFO, and HTTPD streaming [8].

Mopidy

Mopidy is a server-side audio application, written in python, which extends the functionality of MPD. It is available on Debian based Linux platforms, such as Raspbian, via the standard "apt" package manager repositories. In its default configuration, Mopidy acts as a local media server, based on MPD. The advantage of Mopidy however, lies in it's extensibility via available "Extensions". These extensions include Spotify connectivity, allowing users not only access to locally stored audio files, but also to streaming audio.

Volumio

Volumio is a stand alone Linux based operating system, or "distribution", which has been designed specifically for audio playback [9]. It is designed to be used on low powered computers, such as the Raspberry Pi. Playback is accessible via a web application user interface. The playback functionality of Volumio is controlled by an MPD server.

Snapcast

Snapcast is a time synchronized client-server audio player. Snapcast reads audio data from the server device, and, utilising the TCP protocol, sends the data to all connected network client devices. Using the aforementioned audio playback softwares, audio data can be passed to a file, which Snapcast may then read from. Synchronicity is achieved by passing the server's time to all clients, allowing their received data buffers to be played back at the appropriate timing.

5.3 Listener Tracking

Another aspect of the project is the listener tracking and audio routing. There are multiple protocols which may be used in order to determine location of a mobile device. Bluetooth "Beacons" or Access Points are used, sending a packet to the device, the signal strength may be used to calculate and approximate distance [10]. Using filtering techniques, distances may be calculated, using Bluetooth, to an accuracy of approximately 1.8 meters [10].

Location using Wi-Fi may provide greater accuracy, however requires specialised hardware [11]. The 802.11AC Wife standard allows for the use of Beamforming, in which multiple antennas transmit at once, allowing for the targeted transmission of data [12]. Using Angle-of-Arrival (AoA), Time-of-Flight (ToF), and Multiple Signal Classification (MUSIC) algorithms, the distance and direction from the Access Point may be determined [13]. The operations which must be performed are complex, and dependent on the hardware being used. As such, less complex solutions, such as RSSI, may be implemented, however, with these solutions comes a reduction in accuracy.

Bluetooth Beacons

A number of Bluetooth Beacon Specifications are currently accessible for use on embedded Linux platforms. These specifications include AltBeacon [14], from Radius Networks, and Eddystone [15] from Google Beacon Platform. These specifications allow for ranging requests, which provide the requesting device with a distance estimate, based on the received signal strength indication (RSSI) from beacon device to the user device.

802.11MC Wi-Fi

The 802.11MC Wi-Fi protocol provides location determination functionality, using Round Trip Time (RTT). This protocol, if implemented on the embedded Linux platforms, would allow for ranging requests to be performed from Android devices on API level 28 (Android 9 "Pie") [16].

6 Concepts, Modelling, and Design

There are two main design aspects to this project, firstly the audio streaming server, with multiple connected devices, and secondly, the listener tracking. Outlined are the proposed solutions to each of these aspects, along with any complications or limitations faced.

6.1 Audio Streaming

The proposed design solution for the implementation of the audio streaming system utilizes three Raspberry Pi 3 Model B+, each with a DAC for outputting high quality audio. The chosen audio server software runs on one device, alongside the streaming server, with the streaming clients running on the other devices.

Figure 6.1 shows the placement of the client and server devices, with the server being the lower of the devices on the downstairs portion of the floor plan. This device is connected to the network router/access point. Each of the client devices is connected to the network via Wi-Fi.

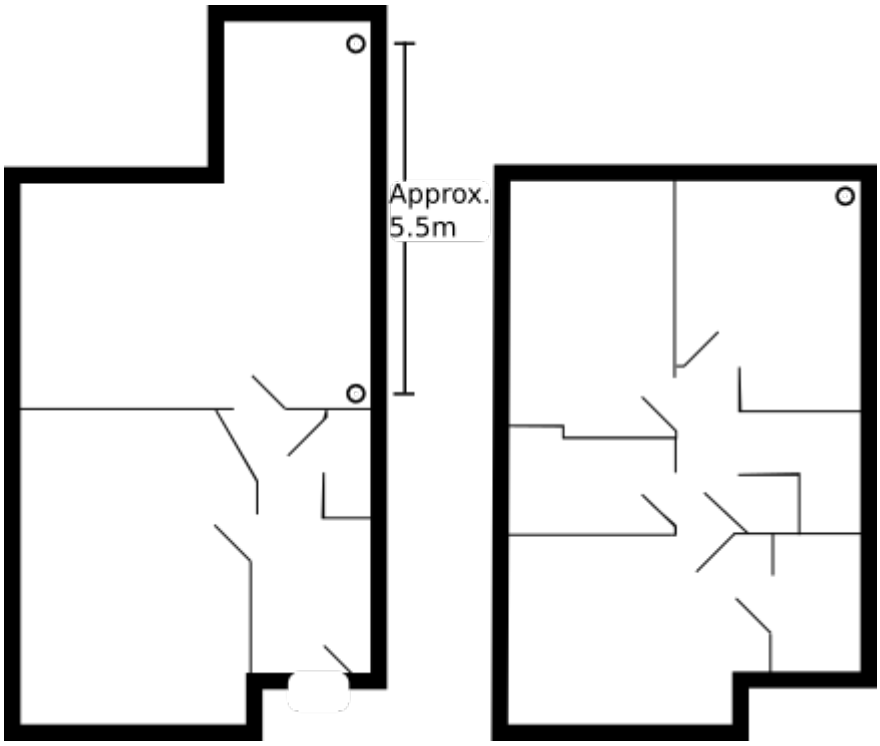

Figure 6.1: Floor Plan with Downstairs (Left), Upstairs (Right), and Raspberry Pi Locations (Represented by 'O's)

Utilising Snapcast, multiple client devices may be connected to a central server device, with audio playback controlled from this server device, and transmit across the network for synchronous playback on the client devices.

6.2 Listener Tracking

Listener tracking is implemented using a combination of Bluetooth Low Energy Beacons, and an accompanying Android application. The BLE Beacons must be available to be scanned from the Android application, which, in turn, must control the volume of the audio client devices,

Beacons

The AltBeacon BLE Beacon protocol is used for proximity data. The audio client devices are configured using modified "BlueZ" example code [17]. AltBeacon is an open-source implementation of Apple's iBeacon standard. The AltBeacon device constantly transmits data, which is intended to be read by the Android application for the purpose of ranging.

While there is an "Android Beacon Library", which allows for ranging of Google's Eddystone Beacons [18], this library is not available in Flutter, which negates the cross-platform compatibility advantage held by Flutter applications. This is discussed further in the following section.

The AltBeacon protocol has a data structure which has a number of significant bytes. Bytes 7 and 8 must be set to "0xBEAC", which represents the AltBeacon advertisement code [19]. Bytes 9-29 are reserved for the unique beacon identifier. In order to correctly calibrate the AltBeacon, the average RSSI value at a distance of 1 meter from the beacon device must be measured, and this value placed in byte 29.

Android Application

The proposed Android application, built using Google's "Flutter" framework, connects to the audio server, allowing for the control of volume parameters. This is achieved via the transmission of JSON encoded messages, as defined in the Snapcast JSON RPC API [20], via a TCP socket connection from the application to the server device, on port 1705. Given an audio client device ID, which may be parsed from a server response message, the client devices volume can be adjusted, or muted from within the application. In order to create a JSON message in Dart, the language used by Flutter, a number of classes and constructors are required. The following is the Snapcast command for adjusting the volume of a connected client device.

Listing 6.1: Example Snapcast "Client.SetVolume" Request

```
{"id":8,
"jsonrpc":"2.0",
"method":"Client.SetVolume",
"params":
    {"id":"00:21:6a:7d:74:fc",
    "volume":{"muted":false,"percent":74}
    }
}
```

As both the "params" and "volume" fields have multiple arguments, they each require separate constructors, as follows:

Listing 6.2: Dart JSON "volume" Field Constructor

```
JSONVolume(bool mute, int percent){
```

```
    this.volume["muted"] = mute;
    this.volume["percent"] = percent;
}
```

Listing 6.3: Dart JSON "params" Field Constructor

```
JSONParams(String id, bool mute, int percent){
    this.params["id"] = id;
    this.params["volume"] = JSONVolume(mute,
    percent).volume;
}
```

Finally, the Dart equivalent of the JSON message can be constructed:

Listing 6.4: Dart JSON Client.SetVolume Constructor

```
ClientVolume(String id, bool mute, int percent){
    this.jsonMessage["id"] = 8;
    this.jsonMessage["jsonrpc"] = "2.0";
    this.jsonMessage["method"] = "Client.SetVolume";
    this.jsonMessage["params"] = JSONParams(id, mute,
    percent).params;
    this.messageString = json.encode(this.jsonMessage)
    + "\r\n";   // '\r\n' required for newline delimited
            // JSON messages
}
```

The encoded message can be transmit to the server using the "Socket.connect(IP, Port)", and "Socket.write(String)" methods. An issue arose however in receiving the JSON response from the Snapcast server, with notification responses being received when audio playback was started or stopped, but no responses being received to the "Server.GetStatus", or "Client.SetVolume" requests. This meant that client ID's could not be parsed from the server response, and, as such, client ID's were required to be hard-coded.

The BLE Beacon ranging is implemented by utilising the Flutter "beacons" plugin, which supports monitoring, and ranging of both iBeacon and AltBeacon Bluetooth Beacons [21]. This plugin's GitHub repository gives an example application, which, when passed a Beacon ID performs a ranging request, printing the RSSI value to the screen. Also provided are methods for ranging while the application is in the background.

Listing 6.5: Flutter Beacons Generic Beacon Ranging

```
Beacons.ranging(region: BeaconRegion(
    identifier: 'test',
        ids: ['id1', 'id2', 'id3'],
      ),
      inBackground: true,
      ).listen((result) {
            final Beacon beacon = result.beacons.first;
      });
```

When combined with the volume control code, the intended implementation is to increase volume as the listener moves closer to the client device, and decrease the volume on the device being moved away from. This implementation was not completed due to issues faced in Dart development, with packages inconsistently loading correctly. This could be due to the relative infancy of the Flutter framework, or due to an inexperience in developing using Flutter and Dart. The advantages of using Flutter however, are that applications, using standard packages, and not platform specific code, may be executed

on both Android and iOS devices, for example, the JSON volume control code could be executed on an iOS device, if available.

All code can be found in the public GitHub at the following location: `https://github.com/mLenehanDCU/ECEIOTProject`

6.3 Extensibility

An ideal solution to the system proposed in this project would provide extensibility, both in the number of clients, and in the number of concurrent users. With the implementation of Snapcast, any number of client devices may be used, so long as the network can handle the number of connections. Many router/access points combinations have a limit on the number of Wi-Fi devices which can be connected to the network at once, with the Sky Q Hub used having a maximum number of 64 connected devices [22].

As devices are added, the accuracy of the BLE beacons could act as an obstruction to extending the number of devices. Differences in accuracy between closely placed beacons could result in the scanning application choosing to route audio to the incorrect client device. However, as Snapcast can group client devices, and modify group volumes rather than individual volumes, client devices placed in too close of a proximity for accurate differentiation could be grouped, with one device in a group acting as a BLE beacon.

Within the context of a home audio setup, it could be desirable to have multiple users have separate access to audio playback, from different client, at the same time. MPD and Mopidy can both be configure to run in multiple instances on the same device, with different ports. From these multiple audio server software instances, audio can be fed to Snapcast, which has the ability to be configured with multiple streams, from multiple inputs, which can be assigned by the user [23].

7 Experimental Equipment and Procedures

7.1 Experimental Equipment Requirements

The following hardware equipment is required for the testing of both the audio server software and the listener tracking implementations:

Experimental Equipment	
Description	**Quantity**
Raspberry Pi 3 Model B+	3
T5989DV - AC/DC Power Supply, 1 Output, 13 W, 5.1 V, 2.4 A	3
Adafruit I2S Audio Bonnet for Raspberry Pi - UDA1334A	3
SanDisk SDSDQAF3-008G-I	3
Audio Amplifier/Speaker Device (with Line In)	3

Table 7.1: Required Experimental Equipment

Audio Server Software Equipment Requirements

In order to follow the testing steps outlined below in Section 7.2, three Raspberry Pi Model 3 B+ are required, along with three T5989DC AC/DC Power Supplies, three SanDisk Industrial 8GB Class 10 MicroSD cards, two Adafruit I2S Audio Bonnets, and two audio amplifier/speaker devices for audio output. This will allow for the client server streaming implementation between two Raspberry Pi's, with the third utilised for monitoring the devices used for streaming.

Listener Tracking Equipment Requirements

In order to follow the testing steps outlined below in Section 7.2, three Raspberry Pi Model 3 B+ are required, along with three T5989DC AC/DC Power Supplies, three SanDisk Industrial 8GB Class 10 Micro SD cards, and one Android device. This will allow for the Bluetooth AltBeacons to be implemented on the Raspberry Pi's, with the Android device running the Bluetooth Beacon detection application.

7.2 Experimental Procedures

Audio Server Software Testing Procedure

The testing steps outlined below will initiate playback of the available audio files from the audio server. Audio will be played for a total of six hours, divided amongst the three available audio formats, WAV, FLAC, and MP3. As shown below, the three available albums

are “Hardwired To Self Destruct” by Metallica, in WAV format, “Superfuzz Bigmuff” by Mudhoney in FLAC format, and “Sonic Highways” by Foo Fighters in MP3.

The Music Player Client “mpc” is used to control playback on each of the audio server softwares. The following mpc commands are used for playback configuration:

- ”add”
  - Adds the specified audio to the current playlist
- ”repeat”
  - Toggles the repeat option, or sets this option to the input argument, i.e. on or off
- ”play”
  - Begins playback of the current playlist
- ”stop”
  - Stops playback of the current playlist
- ”clear”
  - Clears all audio files from the current playlist

Testing Steps:

The following test procedure must be followed in order to retrieve data for the comparison of the chosen audio server softwares.

1. Using ssh, connect to the audio server device
   - $ ssh pi@<Server Pi IP>
2. Create a crontable on the server device to start and stop audio playback at a set time(s):
   - $ crontab -e
   - Enter the following lines to the crontable
     - 00 10 * * * mpc add Hardwired_To–Self–Destruct_BoxSet_WAV/ && mpc repeat on && mpc play
     - 00 12 * * * mpc stop && mpc clear && mpc repeat off
     - 02 12 * * * mpc add mudhoney–superfuzz_bigmuff–flac/ && mpc repeat on && mpc play
     - 02 14 * * * mpc stop && mpc clear && mpc repeat off
     - 04 14 * * * mpc add Sonic_Highways/ && mpc repeat on && mpc play
     - 04 16 * * * mpc stop && mpc clear && mpc repeat off

3. Once testing of all available audio formats is complete, replace the audio server software - ”MPD” - with the ”Mopidy” audio server software. Repeat step one for the ”mopidy” server software.

4. Once testing of all available audio formats is complete, replace the audio server software - ”Mopidy” - with the ”Volumio” audio server software. Repeat step one for the ”Volumio” server software.

5. Record the data output from Munin, available in the browser from <Munin Server IP>/munin.

For analysis, compare all applicable parameters as recorded from Munin. The results and analysis may be seen in Section 9.1.

Listener Tracking Testing Procedure

The testing steps outlined below will test the ranging capabilities of three Bluetooth Beacons, as measured using an Android Smartphone. Three AltBeacons, running on Raspberry Pi 3 Model B+, are scanned for, with ranging requests done at distances of 1-5 meters at one meter intervals.

Nicolas Bridoux's “Beacon Scanner” Android application [24] is used in order to approximate the distance from the Android device to the AltBeacon, utilising Bluetooth RSSI to estimate this distance. The following information may be taken from the application:

- ”Beacon Type”
  - Returns the type of Bluetooth beacon being used, e.g. iBeacon, Eddystone Beacon, and AltBeacon
- ”Beacon ID”
  - The UUID for the beacon being used
- ”RSSI”
  - The Received Signal Strength Indication from the Bluetooth beacon to the Android device
- ”TX”
  - The signal transmission strength from the Bluetooth beacon

Testing Steps:

The following test procedure must be followed in order to retrieve data for the ranging accuracy of the Bluetooth beacons.

1. Using ssh, connect to the Bluetooth beacon device.
   - $ ssh pi@<Beacon Device IP>
2. Execute the AltBeacon script on the Raspberry Pi.

- $ sudo ./altbeacon

3. Enable Bluetooth on the Android device.
4. Run the Beacon Scanner application on the Android device.
5. Place the Android device at a distance of 1m from the beacon device.
6. Begin scanning for Bluetooth devices.
7. Record the estimated distance from the beacon device to the Android device.
8. Repeat steps 5-7 for distances of 2, 3, 4, and 5 meters.

For analysis, compare all distance measured in the Beacon Scanner application with the actual distance. The results and analysis may be seen in Section 9.2.

8 Implementation and Testing

8.1 Audio Server Software Implementation

In order to execute the required testing of the audio server software, the following softwares must be installed on the server device, and the following configurations completed.

SSH

SSH is a protocol used for end-to-end client-server secure connections [25]. SSH allows for remote login, and file transfers, from a client to a server device, i.e. from a development PC to an embedded Linux device. SSH utilises "keys" for authentication, and provides secure encryption between the client and server. Within this project, SSH is used in order to log in to, and run commands on the Raspberry Pi's, which do not have a graphical desktop environment.

Munin

Munin is a server performance monitoring software, which runs on an Apache server, with the client software running on each device requiring monitoring [26]. The recorded information is hosted on a locally accessible website, at the IP address of the server device. The output information is displayed in graphical representation, which can be analysed.

Cron

Cron is a scheduling utility, which allows for the automation of command execution at specified times, or set time intervals [27]. Using a crontable, a file for entering cron jobs, the required testing schedule can be run on the audio server Raspberry Pi. For the purposes of testing the audio server software while streaming audio files of different formats, a crontable is configured to play audio in the WAV format, followed by audio in the FLAC format, followed by audio in the MP3 format. Each audio format is played continuously for two hours, with a two minute space between formats.

MPC

MPC is the "Music Player Controller", a software used for controlling MPD, or MPD derived softwares [28]. As such, it may be used to control MPD, Mopidy, and Volumio. Within the audio server software testing procedure, MPC is used in order to queue, play, and stop audio playback on each of the server configurations.

Configurations

On the MPD server device, the configuration file, found at the location "/etc/mpd.conf" must be modified in order to output audio to the Snapcast server. The default "ALSA" output must first be deleted, and a new "FIFO" output must be added as follows:

Listing 8.1: MPD SnapServer Configuration

```
audio_output {
    type      "fifo"
    name      "my pipe"
    path      "/tmp/snapfifo"
    format    "48000:16:2"
    mixer_type   "software"
}
```

On the Mopidy server device, the configuration file, found at the location "/etc/mopidy/-mopidy.conf" must be modified in order to output audio to the Snapcast server. The default audio output must be removed, with a new "filesink" output added as follows:

Listing 8.2: Mopidy SnapServer Configuration

```
[audio]
output = audioresample ! audioconvert !
audio/x-raw,rate=48000,channels=2,format=S16LE
! wavenc ! filesink location=/tmp/snapfifo
```

These audio outputs are used to feed audio to the Snapcast server, to be sent via the network to the Snapcast client devices. The MPD configuration can also be used on the Volumio server as Volumio utilises MPD and it's configuration files for its audio playback.

8.2 Audio Server Software Testing

A number of parameters must be tested in order to determine the optimal open-source audio server solution. Each audio server software is tested under equal testing conditions, with the values for network usage, CPU temperature, CPU load, and CPU frequency monitored and recorded.

Testing setup consists of three Raspberry Pi's, each running the Raspbian Stretch Light OS, with the exception of the Volumio server test configuration. One Raspberry Pi runs the audio server software, and the Snapcast server software. The second Raspberry Pi runs the Snapcast client software. The final Raspberry Pi runs the Munin server software, allowing to monitor the clients, which are running on the other two Raspberry Pi's. Both the audio server device and the Munin server device are connected to the network router/access point via Ethernet, with the client device connected to the network wirelessly.

Munin returns graphs of system parameters for the system being monitored. The graphs are divided into the categories of "Network", "Processes", "System", and "Sensors". The information contained in these graphs must be extracted, with the maximum, minimum, and average parameter values placed in a table for analysis.

8.3 Listener Tracking Implementation

In order to execute the required testing of the listener tracking, the following softwares must be installed, and the following configurations completed on the beacon devices, and the Android device.

SSH

As described in Section 8.1, SSH allows for the remote login from a client device to a server device, allowing for commands to be run on the Raspberry Pi, sent from a client PC.

BlueZ

BlueZ is the "Official Linux Bluetooth protocol stack" [29]. BlueZ provides a number of code examples for creating Bluetooth beacons, using the AltBeacon protocol. These examples can be modified for the creation of AltBeacons with unique ID's, and allow for the calibration of these beacons.

Beacon Scanner Application

The Beacon Scanner android application allows for the monitoring and ranging of Bluetooth beacons from an android device [24]. Utilising Bluetooth RSSI, the application estimates the distance from the Android device to the Bluetooth beacon. The device reports on beacon type, ID, RSSI, and Transmission power. These parameters allow for the identification and calibration of the beacon, along with the location determination for the client.

8.4 Listener Tracking Testing

In order to determine the accuracy and reliability of a Bluetooth Beacon, specifically the AltBeacon protocol, for the purposes of distance measurements in client tracking applications, a number of readings must be taken, at multiple distances from the beacon. This will show how the measured distance value changes with an increase in physical distance from beacon to scanning device.

Testing setup consists of three Raspberry Pi's, each running the Raspbian Stretch Light OS, with an AltBeacon with a unique ID. The AltBeacons must be calibrated, by taking the RSSI value at 1 meter from the beacon device. The value is added into byte 29 of the AltBeacon, allowing for the beacon to more accurately provide ranging information. An Android device running the "Beacon Scanner" application is used to measure the RSSI value, and in turn to calculate an approximate distance to the beacon.

The distance results obtained from the execution of the testing steps outlined in Section 7.2 must be utilised to create tables of physical distance, estimated distance, and the deviation of the estimate from the actual distance. As such, these tables may be used in the analysis of the accuracy of Bluetooth beacons in location determination.

9 Results and Analysis

The following results and analysis have been completed following testing of the audio server softwares, and the client tracking, as described in Section 7.2.

9.1 Audio Server Software

The following tables have been extracted from the data collected from Munin. A full list of results can be found within the Appendices Section 13.2. The client and server data has been combined under the headings of Network, System, and Sensors, for each of the audio server software solutions.

MPD

The network information for the MPD Server and SnapClient configuration below shows there are no Ethernet errors or traffic on the Client device, as it is connected to the network via Wi-Fi. Conversely, on the Server device, there are Wireless network errors, and traffic values, as the server device is both setup on the network via Wi-Fi and Ethernet.

Network						
Eth0 Errors (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Errors (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	830.03m	0.00	835.49m	0.00	853.19m	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	1.25k	1.31k	23.66k	951.47k	29.07k	1.27M
Wlan0 Errors (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	270.53m	0.00	302.72m	0.00	343.80m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Errors (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	327.00m	0.00	335.22m	0.00	349.93m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	669.92	1.58k	939.62k	31.80k	1.26M	39.03k
Wlan0 Traffic (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	186.42	17.12	321.71	22.45	741.32	39.21

Table 9.1: MPD Server and SnapClient Device Network Parameters

Within the Munin system measurements, it can be seen that on the Server device, approximately 964MB of the 1GB of DDR2 RAM on the Raspberry Pi Model 3B+ is in use on average, with average system load of 0.41, and CPU usage of 5.89% (idling at 92% - Note: The Munin monitoring software measures CPU usage percentage from 0-400%, i.e. usage on each CPU core). On the Client device, approximately 327MB of the 1GB of DDR2

RAM is in use on average, with average system load of 0.13, and CPU usage of 3.35% (idling at 96%)

System			
Load Average (Client)			
	Min	Avg	Max
Load	0.02	0.13	0.46
Load Average (Server)			
	Min	Avg	Max
Load	0.03	0.41	1.03
Memory Usage (Bytes) (Client)			
	Min	Avg	Max
Active	164.04M	165.38M	167.67M
Inactive	51.14M	51.17M	51.21M
Unused	670.26M	673.18M	675.15M
Memory Usage (Bytes) (Server)			
	Min	Avg	Max
Active	270.40M	408.59M	437.63M
Inactive	424.17M	541.23M	589.36M
Unused	31.03M	35.88M	43.78M
CPU Usage (%) (Client)			
	Min	Avg	Max
System	1.05	3.35	9.27
Idle	381.73	384.41	394.69
CPU Usage (%) (Server)			
	Min	Avg	Max
System	1.21	5.89	15.08
Idle	363.71	369.60	394.16

Table 9.2: MPD Server and SnapClient Device System Parameters

The Raspberry Pi CPU frequency and temperature were measured using a Munin plugin. The Client device kept an average frequency of 600MHz, with average frequency scaling of 618.10MHz on CPU core 1 and 2, and average frequency scaling of 610.10MHz on CPU core 3 and 4. The average temperature of the Client device is 42.66 °C.

The Server device had an average frequency of 656.53MHz, however at times reached its maximum frequency of 1.4GHz. On CPU cores 1-4 the average frequency scaling is 691.30MHz, and had an average temperature of 56.11 °C.

Sensors			
CPU Frequency (MHz) (Client)			
	Min	Avg	Max
CPU	600.00	600.00	600.00
CPU Frequency (MHz) (Server)			
	Min	Avg	Max
CPU	600.00	656.53	1.40k
CPU Frequency Scaling (MHz) (Client)			
	Min	Avg	Max
CPU1	613.87	618.10	620.98
CPU2	613.92	618.10	620.97
CPU3	613.87	610.10	620.99
CPU4	613.92	610.10	620.98
CPU Frequency Scaling (MHz) (Server)			
	Min	Avg	Max
CPU1	624.32	691.30	842.15
CPU2	624.32	691.30	842.12
CPU3	624.30	691.30	842.14
CPU4	624.30	691.30	842.11
CPU Temperature (°C) (Client)			
	Min	Avg	Max
CPU	41.86	42.66	44.00
CPU Temperature (°C) (Server)			
	Min	Avg	Max
CPU	53.69	56.11	58.52

Table 9.3: MPD Server and SnapClient Device Sensor Parameters

Mopidy

The network information for the Mopidy Server and SnapClient configuration below, again, shows that there are no Ethernet errors or traffic on the Client device, due to the network connection being wireless. The Server device has both wireless and wired errors and traffic.

Network						
Eth0 Errors (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Errors (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	827.43m	0.00	835.03m	0.00	852.73m	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	1.23k	1.34k	25.65k	961.24k	35.72k	1.34M
Wlan0 Errors (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	313.56m	0.00	330.82m	0.00	344.32m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Errors (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	329.11m	0.00	334.61m	0.00	347.44m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	181.87	17.17	280.87	22.18	756.88	31.00
Wlan0 Traffic (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	636.24	1.58k	950.23k	34.74k	1.33M	48.40k

Table 9.4: Mopidy Server and SnapClient Device Network Parameters

In the system measurements, the Server device uses approximately 950MB of RAM on average, with average system load on 0.23, and average CPU Usage of 3.64%. The Client device uses on average approximately 338MB of RAM, with average system load of 0.11, and CPU usage of 3.43% on average.

System			
Load Average (Client)			
	Min	Avg	Max
Load	0.02	0.11	0.26
Load Average (Server)			
	Min	Avg	Max
Load	0.05	0.23	0.40
Memory Usage (Bytes) (Client)			
	Min	Avg	Max
Active	170.56M	171.03M	171.55M
Inactive	56.72M	56.77M	56.81M
Unused	661.03M	661.75M	662.69M
Memory Usage (Bytes) (Server)			
	Min	Avg	Max
Active	161.88M	375.25M	445.90M
Inactive	245.70M	462.17M	701.93M
Unused	26.69M	54.62M	472.22M
CPU Usage (%) (Client)			
	Min	Avg	Max
System	1.01	3.43	9.45
Idle	381.26	384.85	394.51
CPU Usage (%) (Server)			
	Min	Avg	Max
System	1.68	3.64	7.53
Idle	351.52	365.23	386.77

Table 9.5: Mopidy Server and SnapClient Device System Parameters

The CPU frequency of the Client device is on average 608.89MHz, with an average of 616.29 frequency scaling on all four CPU cores. The CPU temperature on the Client device has an average value of 44.63 °C.

The CPU frequency of the Server device kept, on average, at the maximum frequency of 1.4GHz, with all four CPU cores frequency scaling at 646.91MHz, and average temperature of 58.94 °C.

Sensors			
CPU Frequency (MHz) (Client)			
	Min	Avg	Max
CPU	600.00	608.89	1.37k
CPU Frequency (MHz) (Server)			
	Min	Avg	Max
CPU	1.40k	1.40k	1.40k
CPU Frequency Scaling (MHz) (Client)			
	Min	Avg	Max
CPU1	611.73	616.29	620.02
CPU2	611.73	616.29	620.02
CPU3	611.73	616.29	620.02
CPU4	611.73	616.29	620.02
CPU Frequency Scaling (MHz) (Server)			
	Min	Avg	Max
CPU1	638.58	646.91	674.50
CPU2	638.58	646.91	674.50
CPU3	638.58	646.91	674.50
CPU4	638.58	646.91	674.50
CPU Temperature (°C) (Client)			
	Min	Avg	Max
CPU	43.48	44.63	46.14
CPU Temperature (°C) (Server)			
	Min	Avg	Max
CPU	56.93	58.94	60.15

Table 9.6: Mopidy SnapClient Device Sensor Parameters

Volumio

The network information for the Volumio Server and SnapClient configuration below, again, shows that there are no Ethernet errors or traffic on the Client device, due to the network connection being wireless. The Server device has both wireless and wired errors and traffic.

Network						
Eth0 Errors (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Errors (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	826.87m	0.00	835.74m	0.00	856.40m	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	15.75k	1.75k	38.40k	1.01M	44.88k	1.34M
Wlan0 Errors (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	304.44m	0.00	332.58m	0.00	342.16m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Errors (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic (Client)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	631.61	1.60k	993.84k	31.64k	1.32M	39.29k
Wlan0 Traffic (Server)						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	2.30	3.69	13.02	20.92

Table 9.7: Volumio Server and SnapClient Device Network Parameters

Within the System measurements, it can be seen that the Server device uses on average 965MB of the 1GB of available RAM, has average load of 0.11. The Server device has approximately 2.61% CPU Usage on average.

The Client device uses, on average, 224MB of the available 1GB of RAM, and has system

load of 0.14 on average. The Client device averages approximately 4.07% CPU Usage.

System			
Load Average (Client)			
	Min	Avg	Max
Load	0.01	0.14	0.34
Load Average (Server)			
	Min	Avg	Max
Load	0.02	0.11	0.23
Memory Usage (Bytes) (Client)			
	Min	Avg	Max
Active	92.55M	93.46M	94.27M
Inactive	30.49M	30.54M	30.58M
Unused	774.41M	775.71M	777.38M
Memory Usage (Bytes) (Server)			
	Min	Avg	Max
Active	508.26M	525.56M	596.66M
Inactive	296.90M	364.93M	374.74M
Unused	30.66M	36.30M	47.90M
CPU Usage (%) (Client)			
	Min	Avg	Max
System	1.14	4.07	10.54
Idle	381.29	384.25	394.36
CPU Usage (%) (Server)			
	Min	Avg	Max
System	1.50	2.61	3.06
Idle	378.43	382.80	395.45

Table 9.8: Volumio Server and SnapClient Device System Parameters

Within Munins Sensor measurements, it can be seen that the average CPU frequency of the Client device is 1.39GHz with average frequency scaling of 627.54MHz on all four CPU cores. The client device had an average temperature of 45.65 °C.

The Server device's CPU frequency kept at it's maximum value of 1.4GHz, with the average frequency scaling on all four CPU cores also averaging at 1.4GHz. The CPU temperature of the Server device averaged at 59.97 °C.

Sensors			
CPU Frequency (MHz) (Client)			
	Min	Avg	Max
CPU	613.33	1.39k	1.40k
CPU Frequency (MHz) (Server)			
	Min	Avg	Max
CPU	1.40k	1.40k	1.40k
CPU Frequency Scaling (MHz) (Client)			
	Min	Avg	Max
CPU1	621.70	627.54	631.72
CPU2	621.70	627.54	631.72
CPU3	621.72	627.54	631.72
CPU4	621.72	627.54	631.70
CPU Frequency Scaling (MHz) (Server)			
	Min	Avg	Max
CPU1	1.40k	1.40k	1.40k
CPU2	1.40k	1.40k	1.40k
CPU3	1.40k	1.40k	1.40k
CPU4	1.40k	1.40k	1.40k
CPU Temperature (°C)(Client)			
	Min	Avg	Max
CPU	44.55	45.65	47.23
CPU Temperature (°C) (Server)			
	Min	Avg	Max
CPU	58.53	59.97	60.69

Table 9.9: Volumio Server and SnapClient Device Sensor Parameters

Audio Server Software Analysis

From the results in Table 9.1, Table 9.4, and Table 9.7 it can be seen that the Mopidy Audio Server experienced the least Ethernet drops, and the most outbound Ethernet Traffic (with the same value as Volumio). The Mopidy server also had the highest WLAN Traffic. The difference in Ethernet drops experienced by each of the server options is minimal, with the MPD server having an average value of $835.49m$, the Mopidy server having an average value of $835.03m$, and the Volumio having an average value of $835.74m$. There is variation in the output packets from the Ethernet traffic measurements between each Server. The MPD server has an average value of $951.47k$ packets sent, the Mopidy server has an average value of $961.24k$ packets sent, and the Volumio server has an average value of $1.01M$ packets sent. A large difference can be seen in the WLAN traffic between the server options. The MPD server has an average value of 22.45 packets sent, the Mopidy server has an average value of $34.74k$ packets sent, and the Volumio server has an average value of 3.69 packets sent.

From the results in Table 9.2, Table 9.5, and Table 9.8, it can be seen that the highest system load was experienced by the MPD audio server, with an average value of 0.41, and a maximum value of 1.03. The Mopidy and Volumio Server had an average load of 0.23 and 0.11 respectively, with maximum values of 0.40 and 0.23 respectively. The memory usage of the Mopidy server was the lowest, with an average value of $945.38MB$, and a maximum usage of approximately $973.31MB$. The MPD and Volumio Servers had an

average memory usage of 964.12MB and 963.70MB, with maximum values of 968.97MB and 969.34MB respectively. The Volumio Server experienced the lowest CPU usage, with system usage averaging 2.61%, and a maximum value of 3.06%. The MPD and Mopidy Servers had system CPU Usage values of 5.89% and 3.64%, with maximum values of 15.08% and 7.53% respectively.

From the results in Table 9.3, Table 9.6, and Table 9.9 it can be seen that the lowest average CPU frequency, temperature, and CPU frequency scaling were achieved by the MPD server. This server had an average CPU frequency of 656.53MHz, average CPU frequency scaling of 691.30MHz, and average CPU temperature of 56.11°C. The Mopidy and Volumio Servers had average CPU frequency values of 1.4GHz, with average frequency scaling values of 646.91MHz and 1.4GHz respectively. The low CPU frequency value of the MPD Server device can be attributed to a number of issues, such as thermal throttling, however, due to idling, the device can also lower CPU frequency to 600MHz [30].

While streaming, there were no noticeable audio issues detected. It can be seen in the case of all audio server softwares, Section 13.1, the highest network throughput occurs for files of the WAV format, followed by MP3. The lowest network throughput is experienced by audio in the FLAC format. Audio played from both the audio serving device, and the client device, streamed via Snapcast, with no "popping" or audio distortion experienced. As the performance of audio playback was not varying during testing, and due to the minimal differences between the server softwares, it is concluded that, due to ease of configuration, and lowest memory usage and CPU temperatures, along with comparable Network and System measurement values, the MPD server was chosen as the Server software to be used.

9.2 Listener Tracking

The following tables are the outcome of the testing described in Section 7.2, Listener Tracking Testing Procedure. The distance estimates from the Android device to the three calibrated Raspberry Pi AltBeacons are listed, alongside the actual distance measurements.

Server Device Beacon

Bluetooth Distance Measurements (m)		
Distance Measurement	BLE Distance Estimate	Deviation
1	0.97	0.03
2	2.44	-0.44
3	3.19	-0.19
4	3.62	0.38
5	4.54	0.46

Table 9.10: Server Device Beacon Bluetooth Distance Measurements

As seen in Table 9.10, there is a deviation in the distance estimate of 0.03m at 1m, 0.44m at 2m, 0.19m at 3m, 0.38m at 4m, and 0.46m at 5m.

"Client1" Device Beacon

Bluetooth Distance Measurements (m)		
Distance Measurement	BLE Distance Estimate	Deviation
1	0.98	0.02
2	1.34	0.66
3	2.16	0.84
4	1.35	2.65
5	2.40	2.60

Table 9.11: "Client1" Device Beacon Bluetooth Distance Measurements

As seen in Table 9.11, there is a deviation in the distance estimate of 0.02m at 1m, 0.66m at 2m, 0.84m at 3m, 2.65m at 4m, and 2.60m at 5m.

"Client2" Device Beacon

Bluetooth Distance Measurements (m)		
Distance Measurement	BLE Distance Estimate	Deviation
1	1.16	-0.16
2	2.21	-0.21
3	2.89	0.11
4	4.18	-0.18
5	5.34	-0.34

Table 9.12: "Client2" Device Beacon Bluetooth Distance Measurements

As seen in Table 9.12, there is a deviation in the distance estimate of 0.16m at 1m, 0.21m at 2m, 0.11m at 3m, 0.18m at 4m, and 0.34m at 5m.

From the results obtained from testing, it can be seen that there is a maximum deviation of 2.65 meters, with a minimum deviation of 0.02 meters. There is an average deviation of 0.618 meters. There is no direct correlation between increases in physical distance, and increases in the BLE distance estimate visible from the captured data. More thorough testing would be required in order to more accurately determine the accuracy of the BLE distance estimates. From the obtained results however, it can be seen that a minimum accuracy of within 2 meters may be achieved, which is sufficient for movement within a home environment. This, in addition to the multiple measurements being taken, will allow for greater accuracy, with outlier values being able to be discarded from measurements.

10 Ethics

10.1 Audio Tracks

While there have been questions raised in recent years over the ethical problems with streaming services, and the revenue paid to artists whose music is played via their platforms, there are also a number of issues with audio piracy from users illegally downloading audio tracks. As solutions such as the one provided by this project offer options for both streaming and locally stored audio, these ethical issues must be explored in order to determine the impact that a solution such as this may have.

Streaming Solutions

With the large increase in paying subscribers in audio streaming services, such as Spotify, in the last number of years questions have arisen with regards to the payment received by artists. Spotify, which, as of 2018 [4] is the largest audio streaming service, with a paying subscriber count of approximately 96 million as of Q4 2018 [31], as displayed in Figure 10.1, has been criticized for the payment which the artists on its service receive. More popular artists, who hold a larger percentage of the rights to their music can earn between $0.006 and $0.0084 per stream. Studies have shown that per million plays, an artist generates approximately $7,000 on Spotify, or approximately $1,650 on Pandora [32].

While Spotify and other streaming services may provide users with unprecedented access to audio, it has been noted that the effect it has had on audio track sales could be a detriment to artist income. A paper co-authored by Spotify's Director of Economics has examined the "pro-rata" model used by Spotify to distribute subscriber income, which is based on an artists percentage of total track plays, concluding that a pro-rata model is beneficial to artists over a user-centric model when, in the case of $70 million distributable income, increasing system costs exceed 4.64% [33].

Conversely, a 2017 paper from Digital Media Finland explores the benefits of the user-centric system [34]. This study shows that in the "pro-rata" model, individual user habits are not accounted for, with subscription income distributed across all tracks played. In a "user-centric" model, the subscription income is distributed across all of the tracks played by the user. This is beneficial to artists of less popular genres, giving their listeners the opportunity to support the artists they listen to, without placing their money in a pot for distribution.

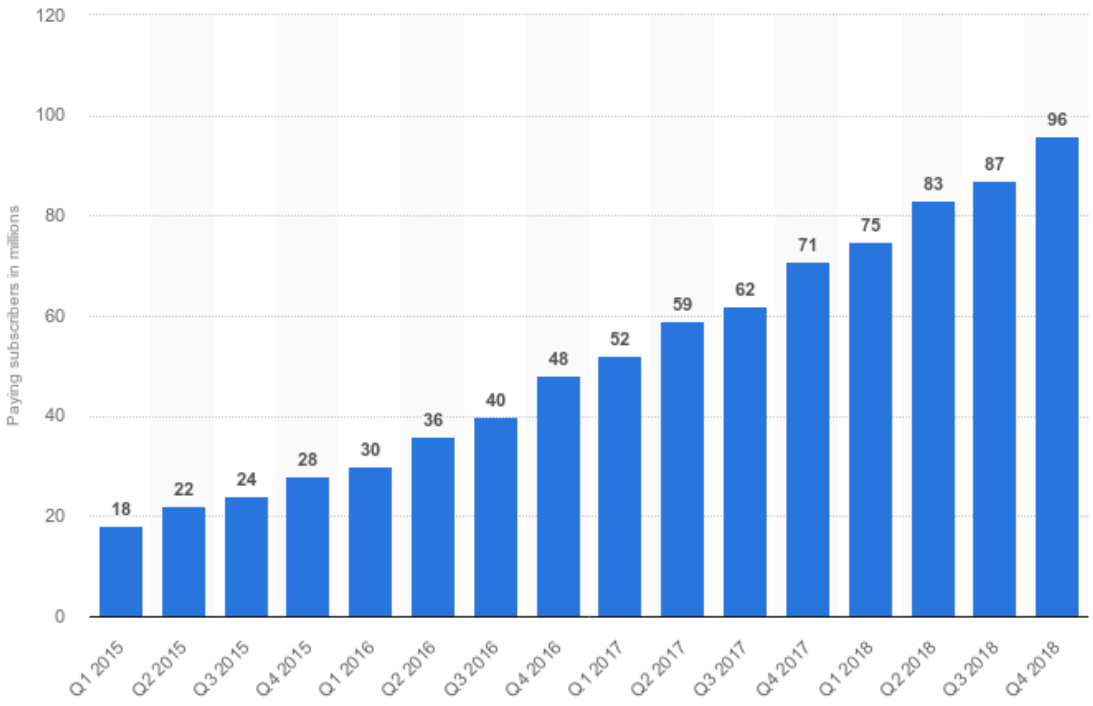

Figure 10.1: Spotify Paying Subscribers in Millions from Q1 2015 to Q4 2018

With the significant increase in spending on streaming service subscriptions over physical media sales, and digital downloads, as in Figure 10.2, the fair and equal payment of artists must be considered. Artists of more niche genres can be negatively affected by the currently used "pro-rata" models, however, the change to a "user-centric" model may not benefit these artists, as they increase the processing costs at the side of the streaming service. As such, care must be taken to ensure that artists incomes are not so negatively affected so as to reduce the number of smaller artists within the industry.

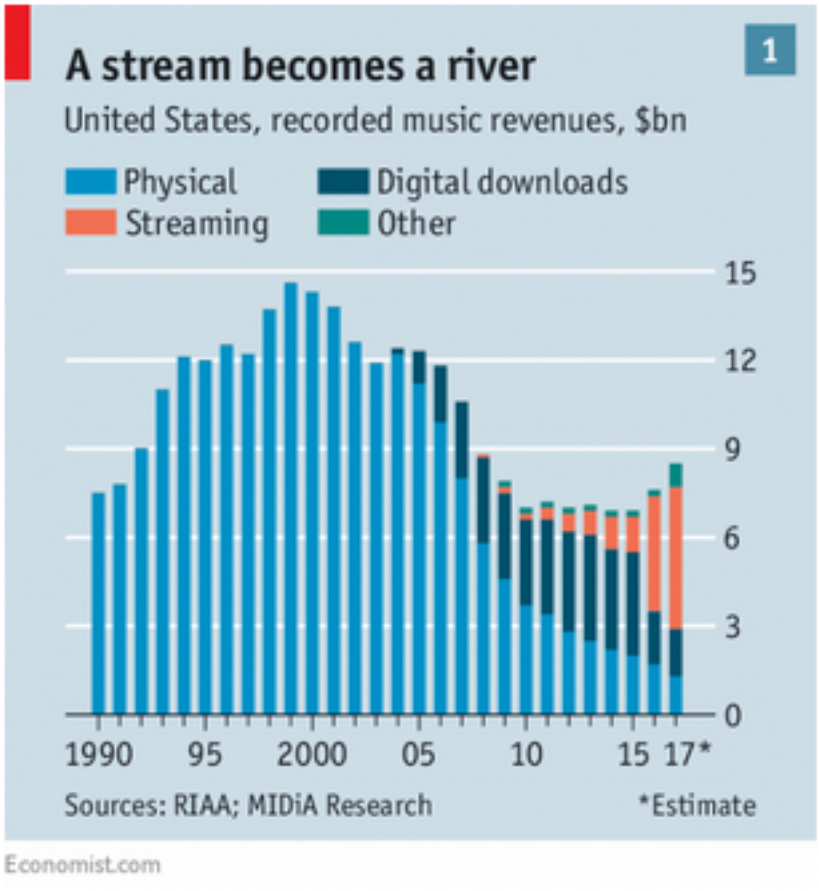

Figure 10.2: Recorded Music Revenues for the United States [4]

Audio Piracy

When considering solutions which use locally stored audio, the ethical issue of sourcing these audio tracks must be discussed. While it may be presumed that, with the increase in the availability of audio streaming, audio piracy would significantly decrease, it has been noted that audio piracy affects physical and digital media consumption [35]. While the Global Online Piracy Study [35] suggests that audio piracy is decreasing, MUSO, a company aiming to study and solve issues with audio piracy, has stated in their annual piracy report a 1.6% increase in global piracy in 2017 [36]. Joost and Quintais, however, contest these figures, noting that MUSO "overestimate the number of and visits to pirate websites" [35].

While piracy via illegal downloads seems to be decreasing, only replacing the declining digital and physical sales [4], illegal streaming via personal licences being used in place of business licences may be a more prominent issue, with a survey showing 83% of small business owners use a personal streaming service account in their premises [37].

Sourcing audio files from legal sources, such as physical or online retailers, is important in order to ensure proper payment is received by artists and production companies. The use of locally stored audio in this project may however increase the use of pirated audio, due to a lack of accessibility to these physical or digital media.

10.2 Listener Tracking

Mobile tracking has a number of ethical implications, imposing on an individuals right to privacy. Access to a users location data must be privileged, accessible only when necessary, and only from the application requiring the data. Bluetooth beacons, as used within this project, have the capacity for two-way communication. As such, there is the concern of the information being intercepted, which could lead to malicious use of the captured data [38].

As proximity based services become more common, it becomes increasingly important to ensure the protection of a users personal data, and location information. Both the Apple App Store, and Google Play Store require that any application which requires access to location services, or to Bluetooth services request permission from the user to access these services. This allows users the ability to block applications accessing this information. Furthermore, laws have been introduced prohibiting companies from using personal data from non-consenting users [39]. Transparency is required throughout the privacy policies of applications and services, in order to correctly inform users of the data being accessed, and the storage and usage conditions of this data.

11 Conclusions

Following the completion of the required testing and analysis for the audio server software, it can be concluded that, although there is little difference between the audio server software with each having advantageous results over the others, the MPD software is the best for system longevity. The software does not cause any significant heat increase within the device while streaming, lowering the risk of wear to the device over prolonged use periods. Mopidy has the advantage of having greater extensibility, allowing for not only the playback of locally stored audio, but also the playback of streamed audio from services such as Spotify. While Volumio is an excellent solution for "headless" audio systems as it is intended to be used, for the purposes of this project, it essentially seen as an MPD server with a graphical user interface. For these reasons, and due to the ease of use and configuration, MPD was chosen as the optimal solution for an audio server software. However, with further configuration, the Mopidy server software could provide greater options for the user.

Testing of the listener tracking using BLE beacons shows that an accuracy of within 2 meters may be achieved, with most measured values being within an accuracy of 1 meter. These measurements, combined with the measurements being taken from multiple BLE beacons ranging estimates, can provide a means for more accurately positioning a user within the space. While the AltBeacon protocol is used in this project, Google's Eddystone beacon protocol could also be used for ranging. However, as the Flutter framework is being used, there is no support for this protocol. This support could be added but would require significant experience in converting Java code implementations to Dart code implementations.

While both the JSON messaging from the Android device to the Snapcast server, and the BLE beacon ranging requests could be implemented, the code could not be combined in order to change volume based on listener location. This could be due to a number of issues, including incompatibilities between the Dart libraries and Flutter plugins being used. As Flutter is a relatively new framework, currently in version 1.2, many of the open-source, community created plugins have issues which may not have been documented at this time. The issues could also be due to inexperience in using the Dart language, which is the primary language used for cross-platform code in Flutter. Java code could be used in order to implement ranging and JSON messaging, however, this would not allow for the code to be used on the iOS platform.

12 Recommendations

The currently proposed implementation can communicate with the Snapcast server, and perform ranging requests with the BLE beacons for location determination. Further testing could allow for the combination of these functionalities. With both code implementations combined in the same application, audio volume on the client devices could be automatically determined via the listeners location.

With the introduction of 802.11MC access points, and Android Pie, the Android Wi-Fi Round Trip Time implementation could be used for more accurate location determination. If the embedded Linux platforms used for the client and server devices had support for this 802.11MC protocol, the same server software implementation could be used, with RTT ranging used in the place of Bluetooth ranging.

The proposed implementation tackles the issues of listener tracking, and high quality audio streaming, however, it does not focus on implementing audio playback control from within the application. Applications such as MPDroid, and M.A.L.P can provide control of the playback features of the audio server softwares [40] [41]. As communication with the audio server is done via TCP sockets, playback controls could be added using a similar implementation as the JSON messaging used for the Snapcast commands.

These recommendations, if implemented within the current system, would allow for greater user control, and improve the accuracy and usability of the application.

13 Appendix

13.1 Audio Server Software Munin Data

MPD

Disk

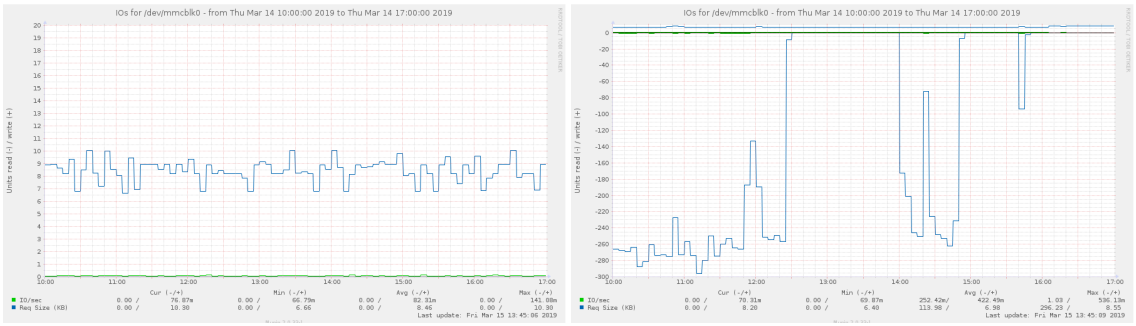

Figure 13.1: MPD Disk I/O on Client and Server Device

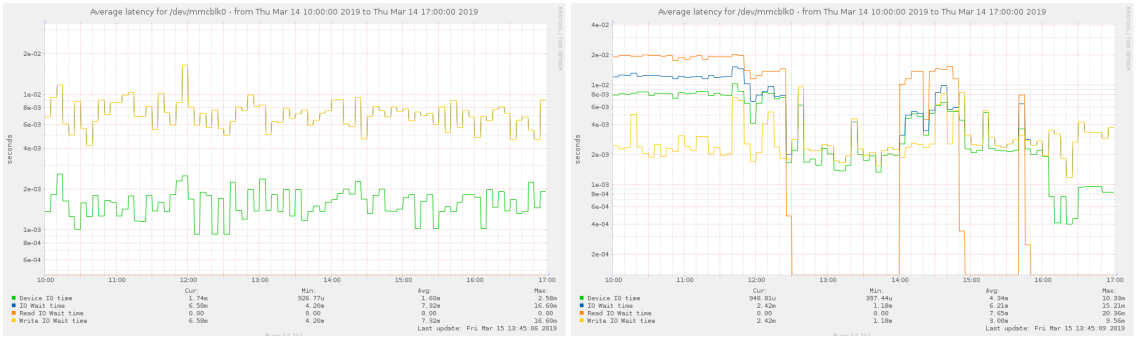

Figure 13.2: MPD Client and Server Device Disk Latency

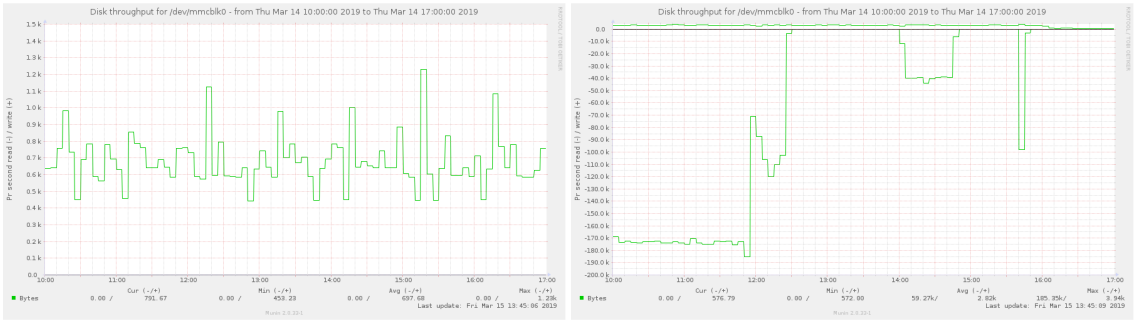

Figure 13.3: MPD Client and Server Device Disk Throughput

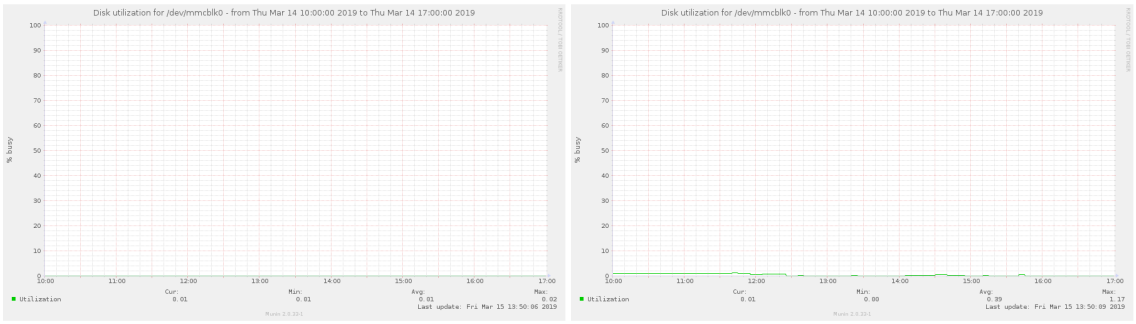

Figure 13.4: MPD Client and Server Device Disk Utilization

Network

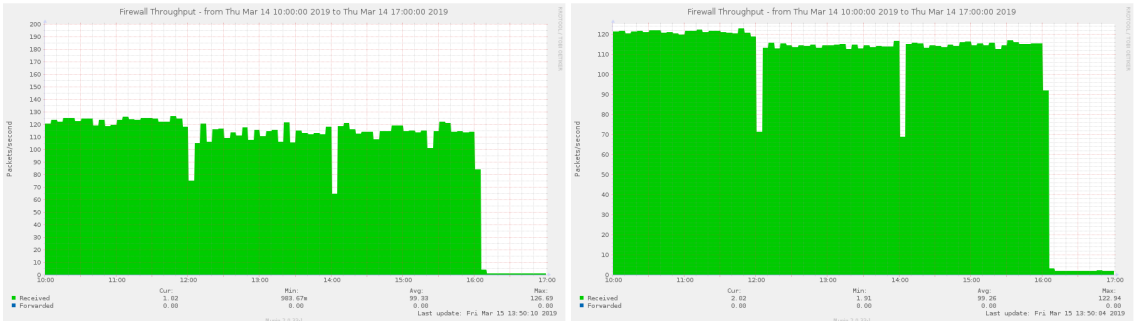

Figure 13.5: MPD Client and Server Device Firewall Throughput

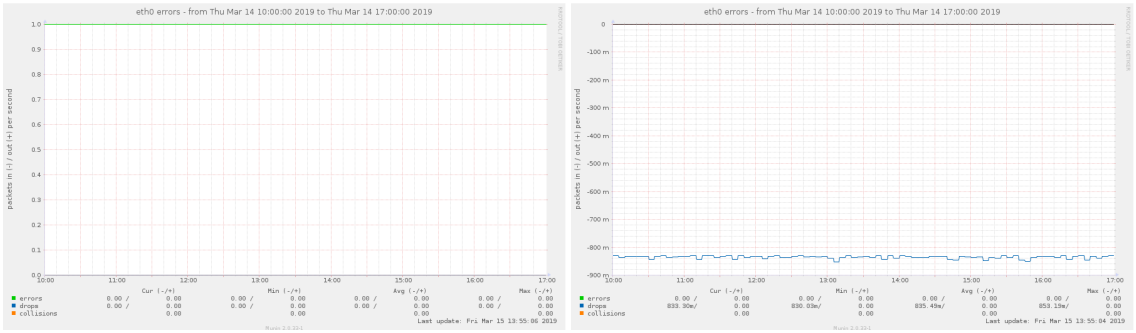

Figure 13.6: MPD Client and Server Device Eth Errors

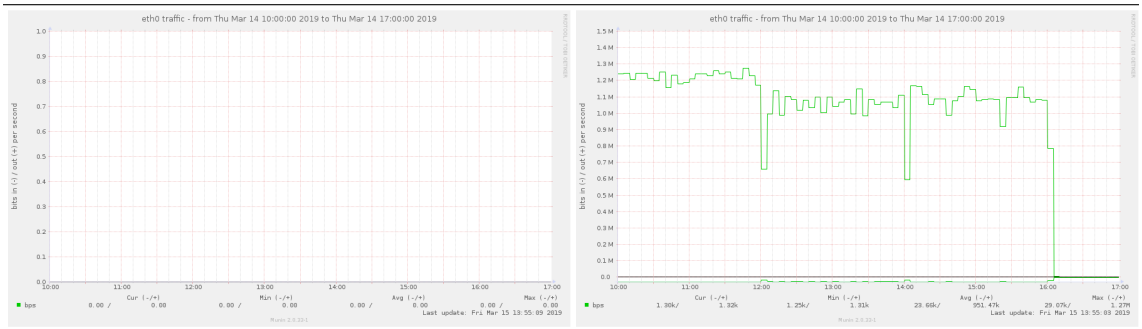

Figure 13.7: MPD Client and Server Device Eth Traffic

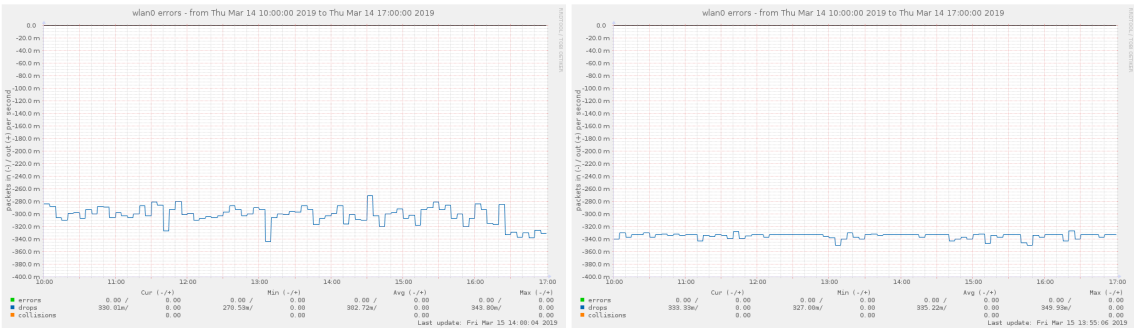

Figure 13.8: MPD Client and Server Device Wlan Errors

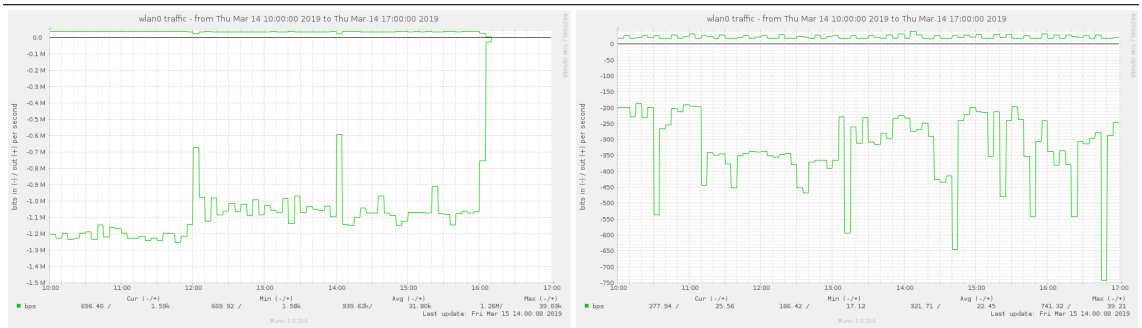

Figure 13.9: MPD Client and Server Device Wlan Traffic

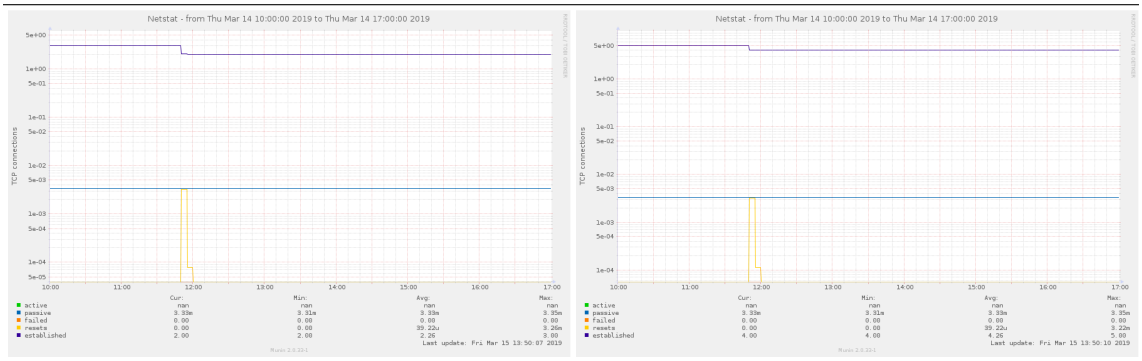

Figure 13.10: MPD Client and Server Device Netstat

Processes

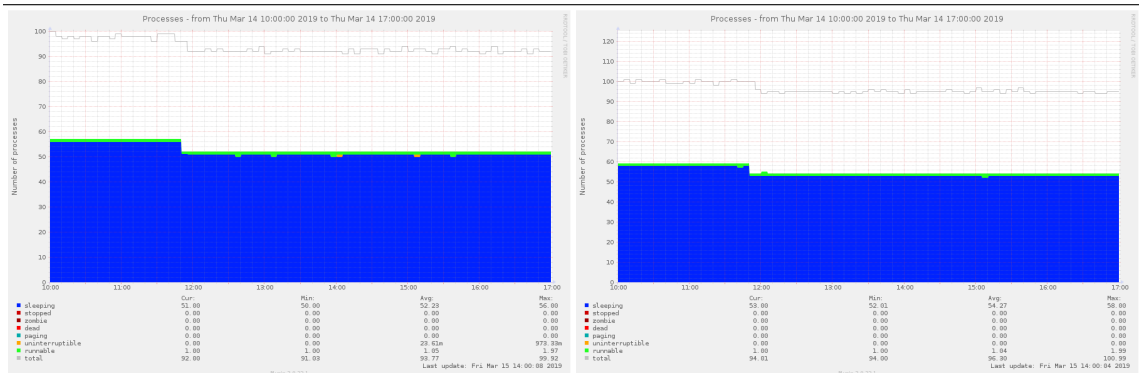

Figure 13.11: MPD Client and Server Device Processes

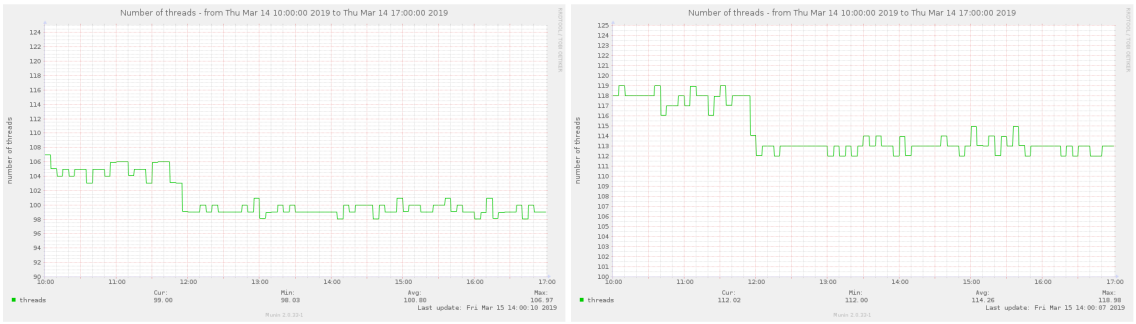

Figure 13.12: MPD Client and Server Device Number of Threads

System

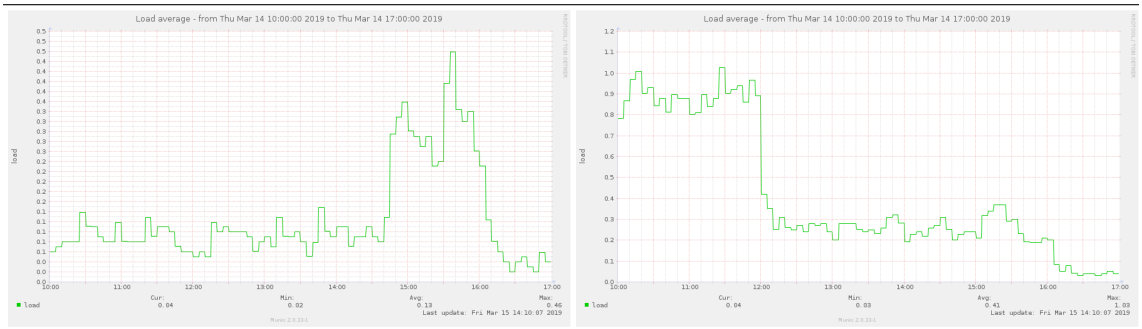

Figure 13.13: MPD Client and Server Device Load Average

Figure 13.14: MPD Client and Server Device Individual Interrupts

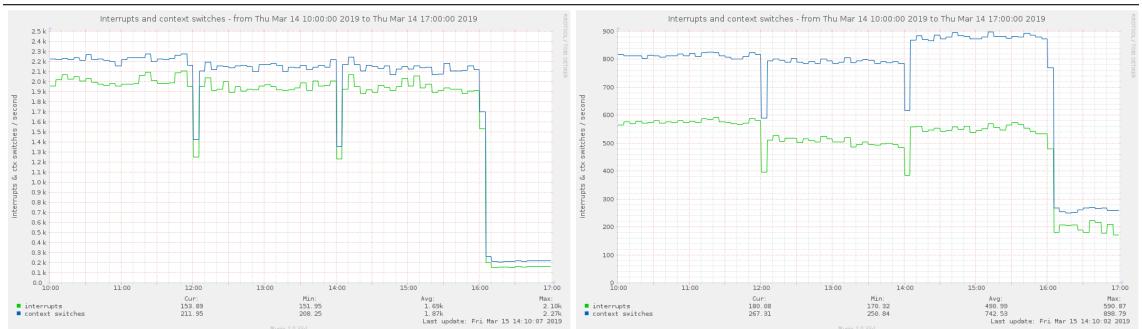

Figure 13.15: MPD Client and Server Device Interrupts and Context Switches

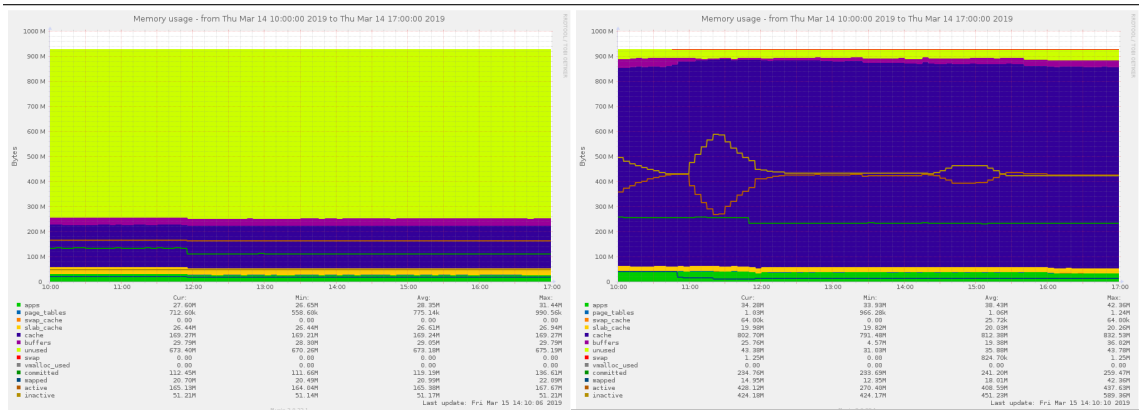

Figure 13.16: MPD Client and Server Device Memory Usage

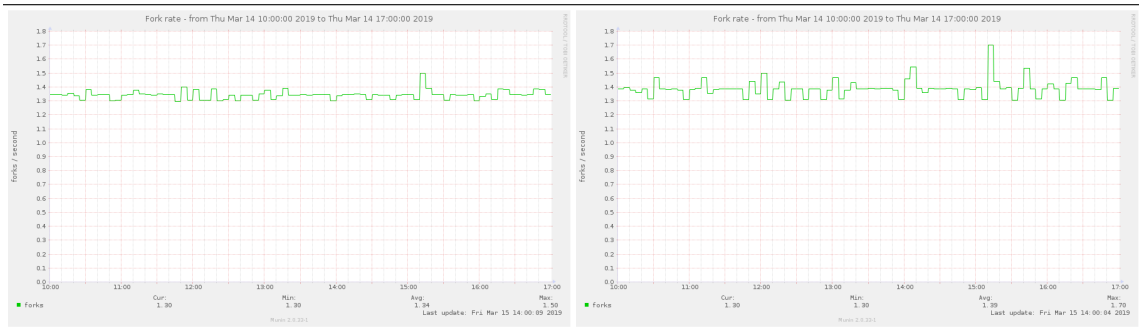

Figure 13.17: MPD Client and Server Device Fork Rate

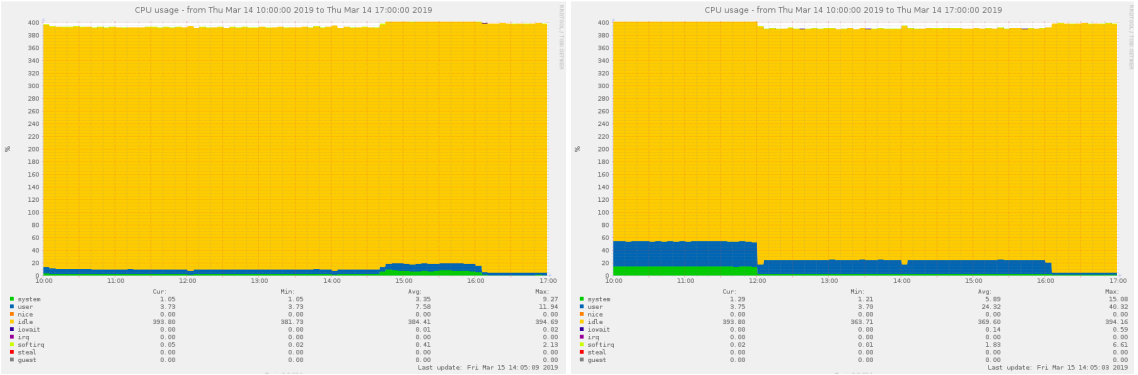

Figure 13.18: MPD Client and Server Device CPU Usage

Sensors

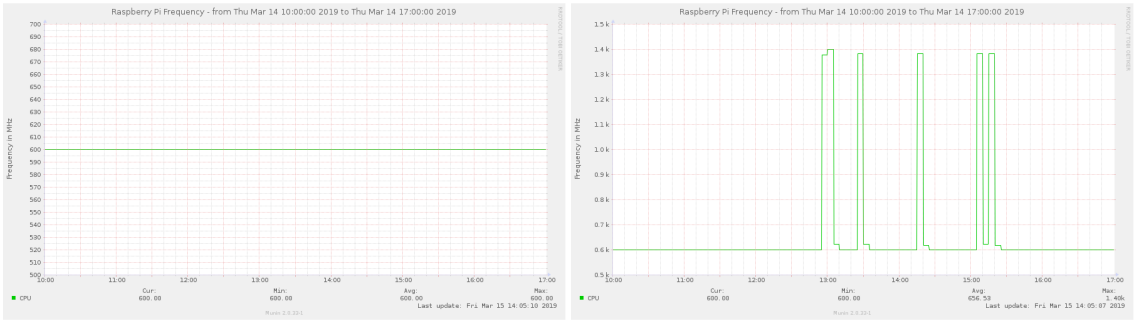

Figure 13.19: MPD Client and Server Device CPU Frequency

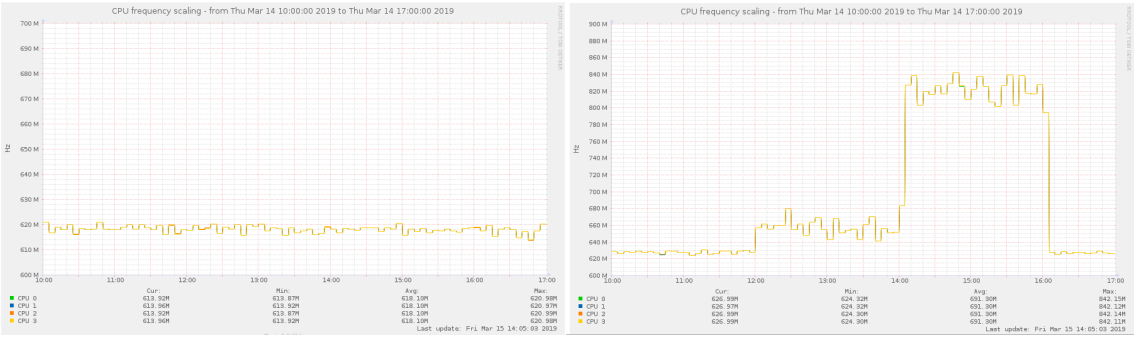

Figure 13.20: MPD Client and Server Device CPU Frequency Scaling

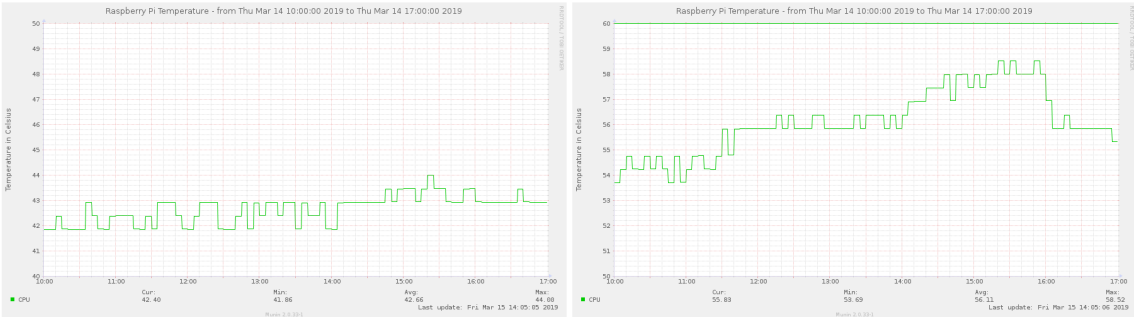

Figure 13.21: MPD Client and Server Device CPU Temperature

Mopidy

Disk

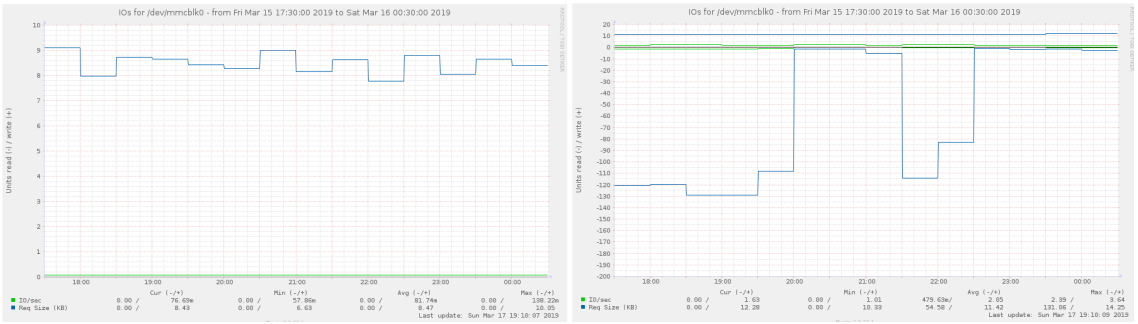

Figure 13.22: Mmopidy Disk I/O on Client and Server Device

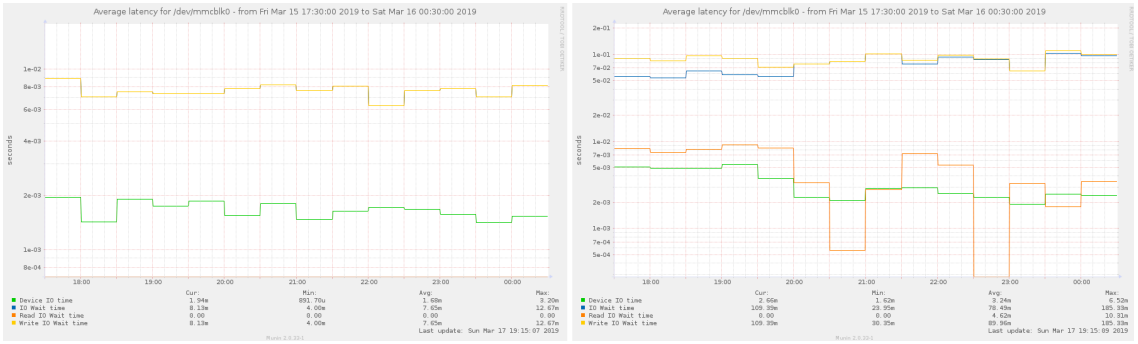

Figure 13.23: Mopidy Client and Server Device Disk Latency

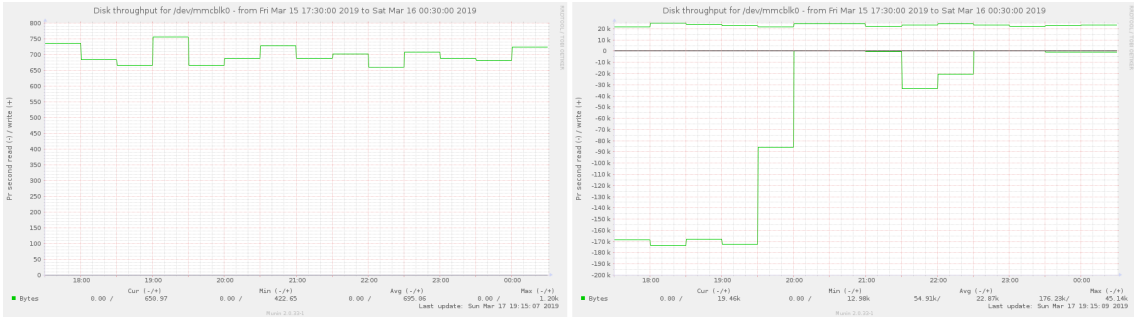

Figure 13.24: Mopidy Client and Server Device Disk Throughput

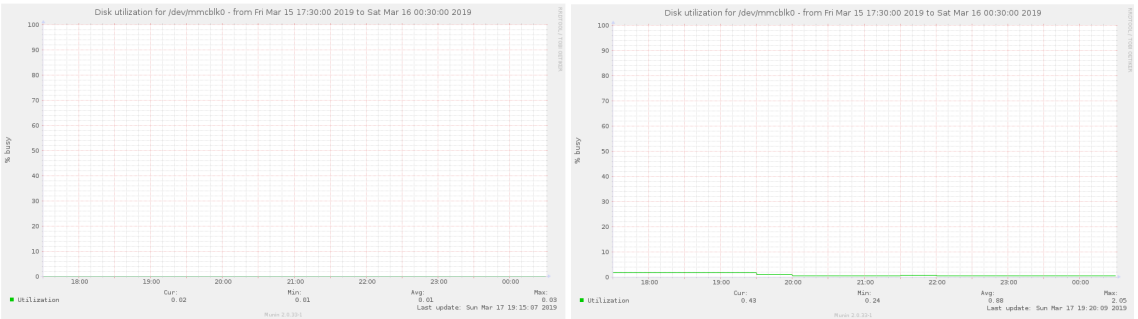

Figure 13.25: Mopidy Client and Server Device Disk Utilization

Network

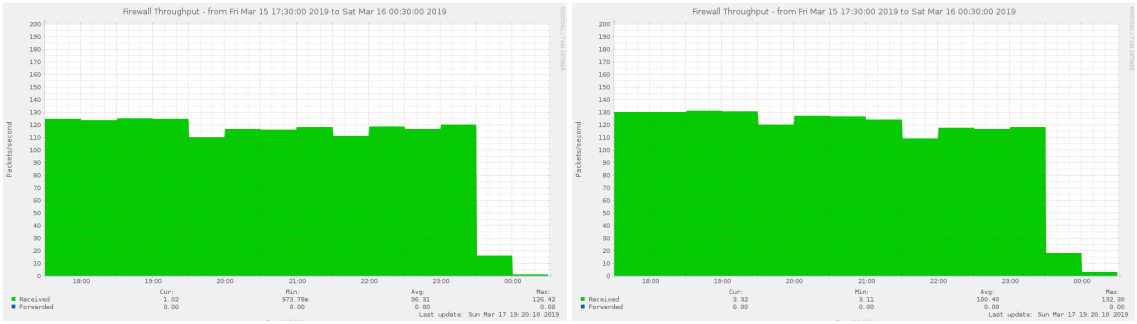

Figure 13.26: Mopidy Client and Server Device Firewall Throughput

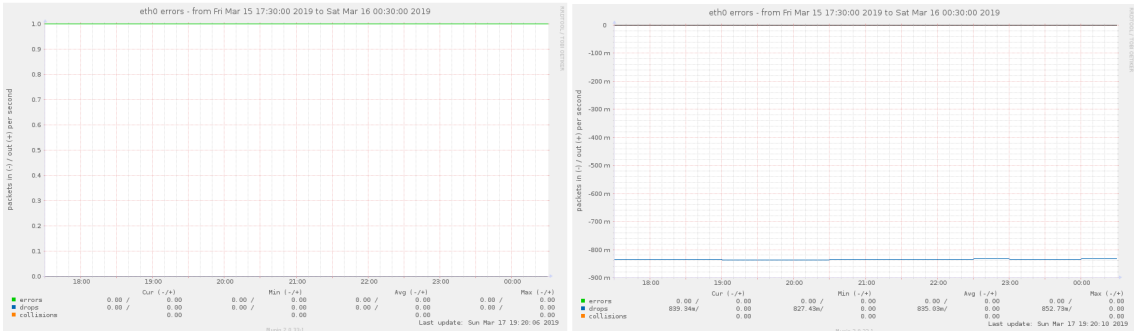

Figure 13.27: Mopidy Client and Server Device Eth Errors

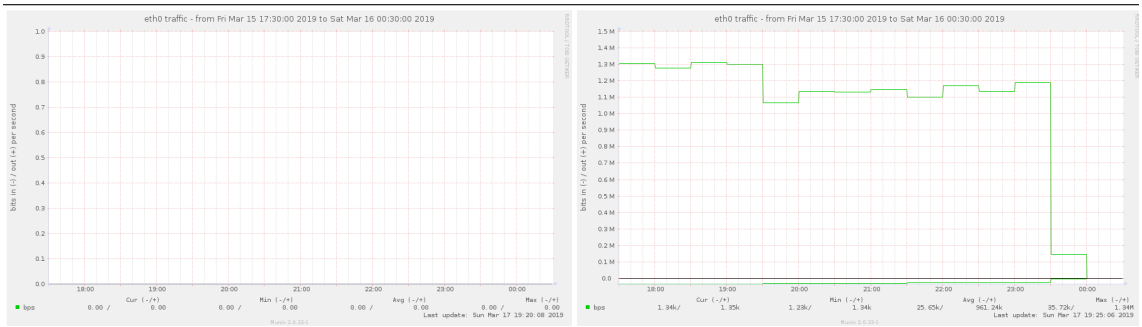

Figure 13.28: Mopidy Client and Server Device Eth Traffic

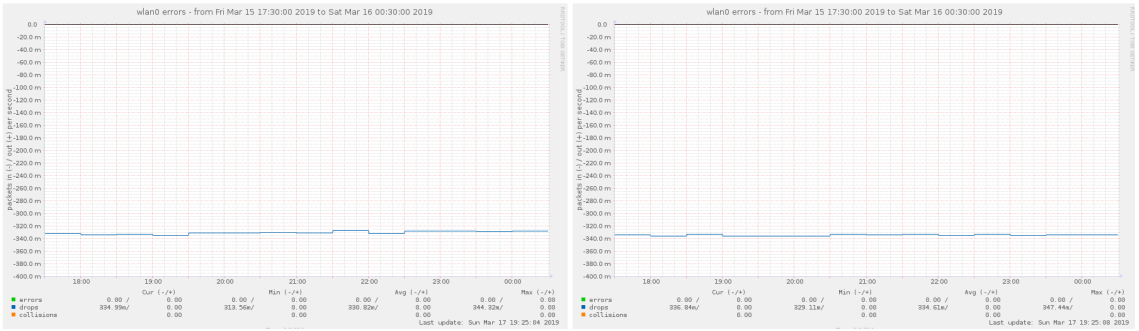

Figure 13.29: Mopidy Client and Server Device Wlan Errors

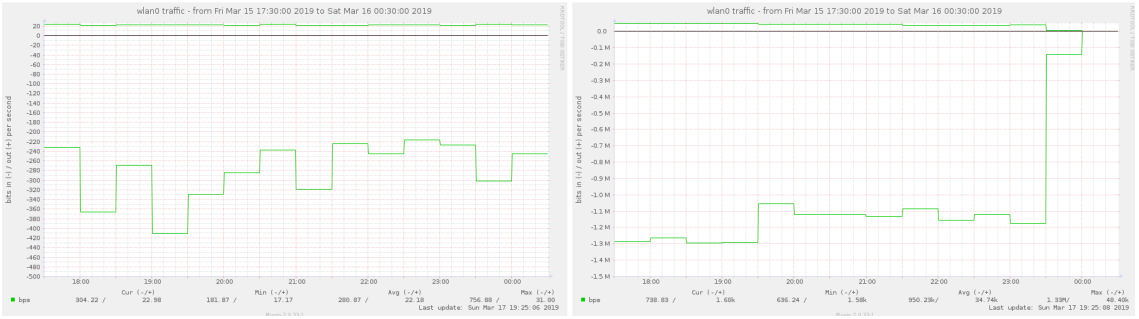

Figure 13.30: Mopidy Client and Server Device Wlan Traffic

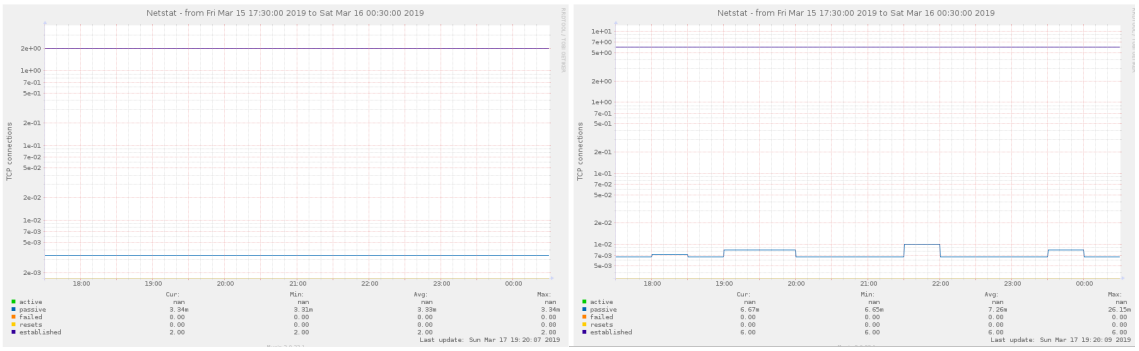

Figure 13.31: Mopidy Client and Server Device Netstat

Processes

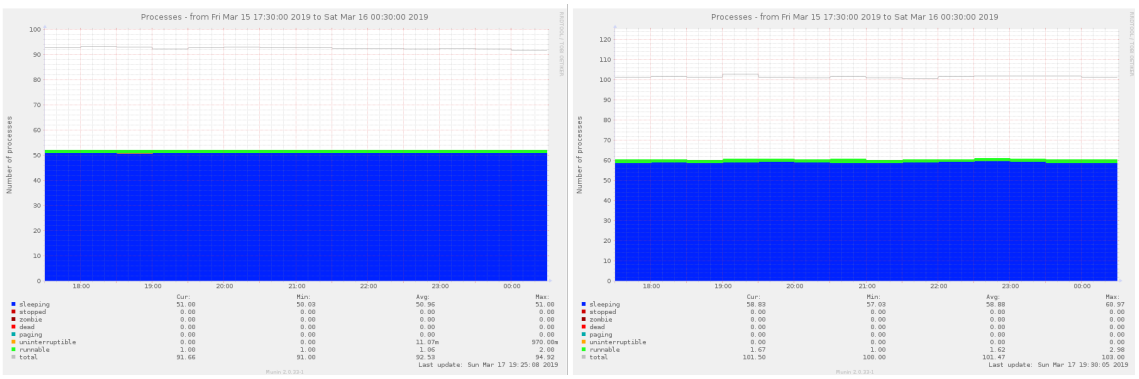

Figure 13.32: Mopidy Client and Server Device Processes

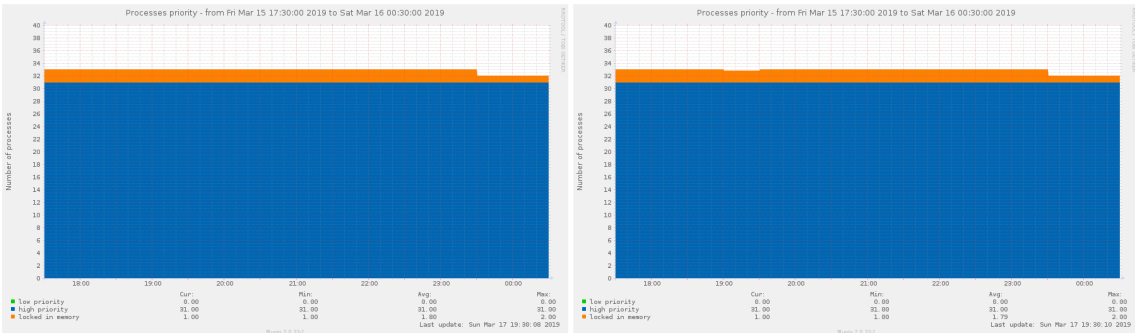

Figure 13.33: Mopidy Client and Server Device Process Priority

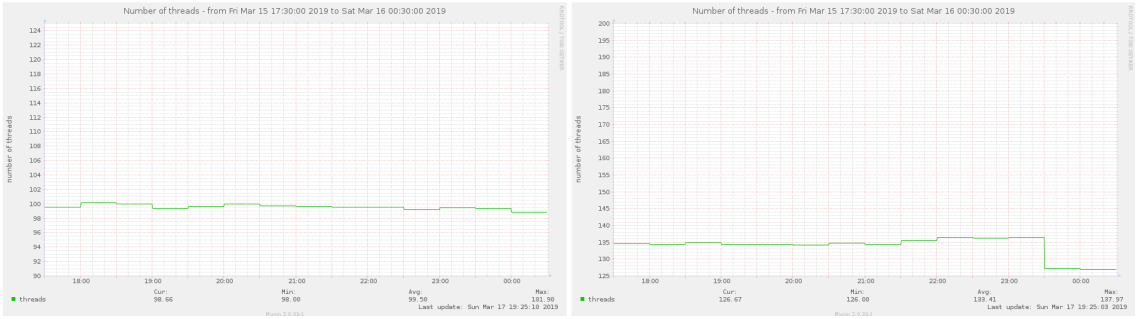

Figure 13.34: Mopidy Client and Server Device Number of Threads

System

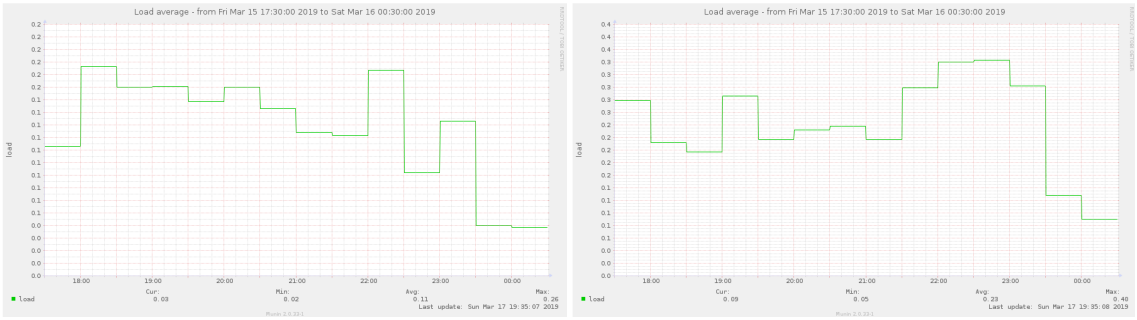

Figure 13.35: Mopidy Client and Server Device Load Average

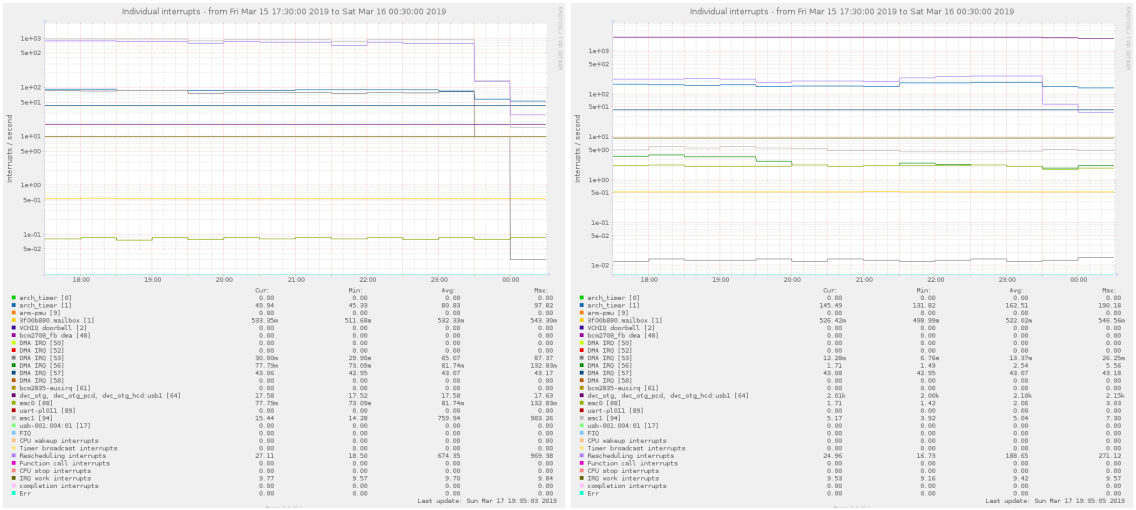

Figure 13.36: Mopidy Client and Server Device Individual Interrupts

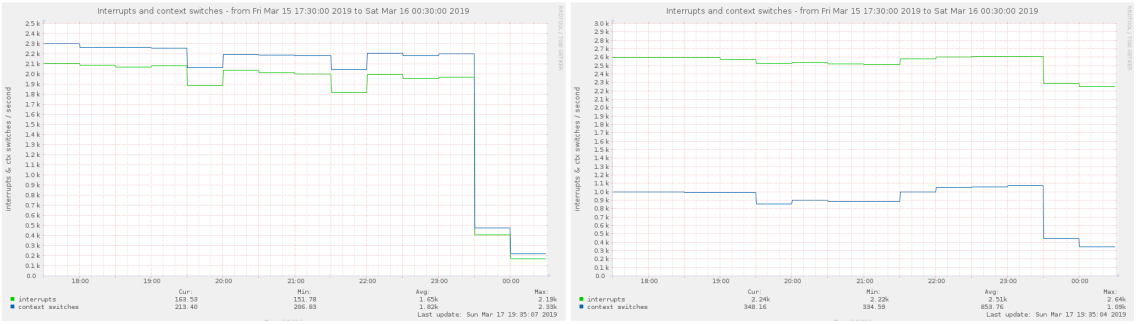

Figure 13.37: Mopidy Client and Server Device Interrupts and Context Switches

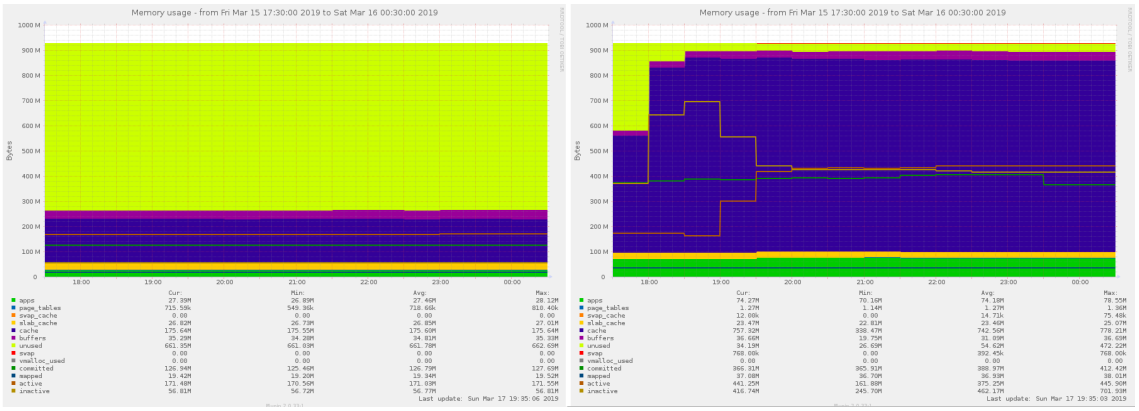

Figure 13.38: Mopidy Client and Server Device Memory Usage

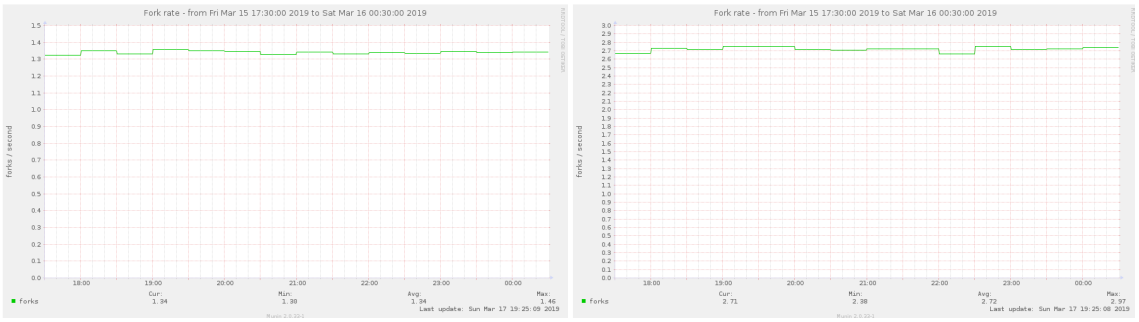

Figure 13.39: Mopidy Client and Server Device Fork Rate

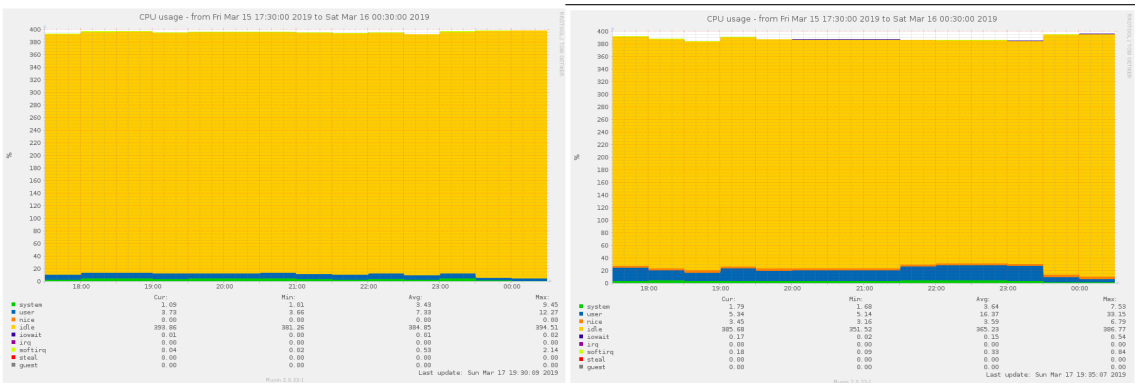

Figure 13.40: Mopidy Client and Server Device CPU Usage

Sensors

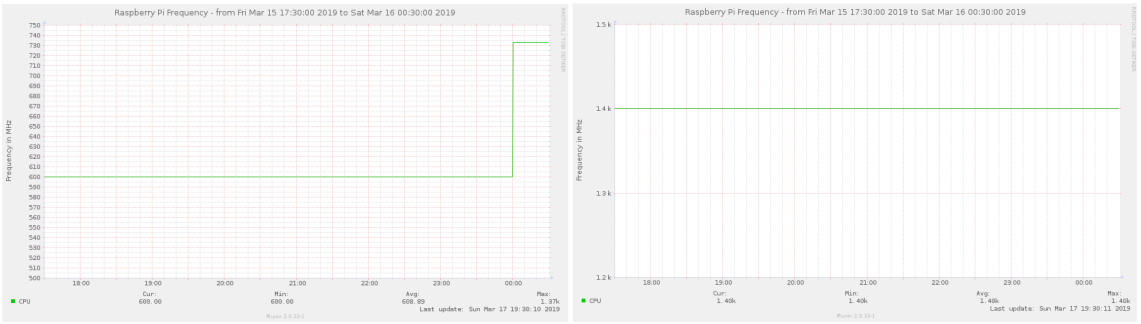

Figure 13.41: Mopidy Client and Server Device CPU Frequency

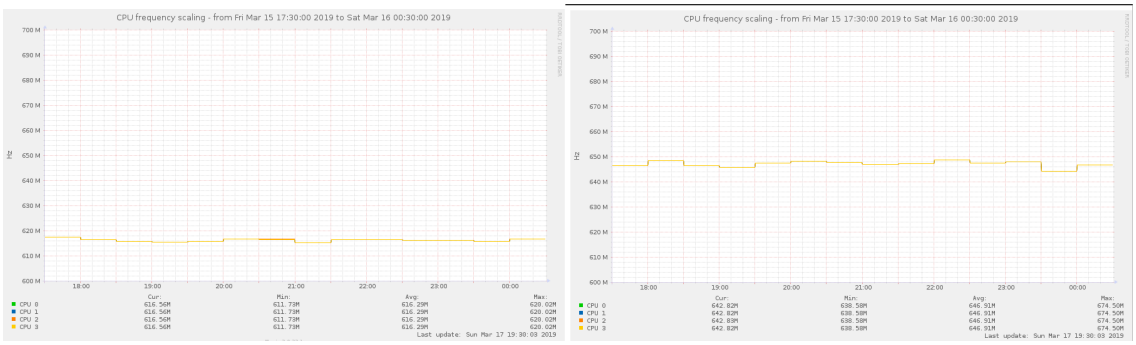

Figure 13.42: Mopidy Client and Server Device CPU Frequency Scaling

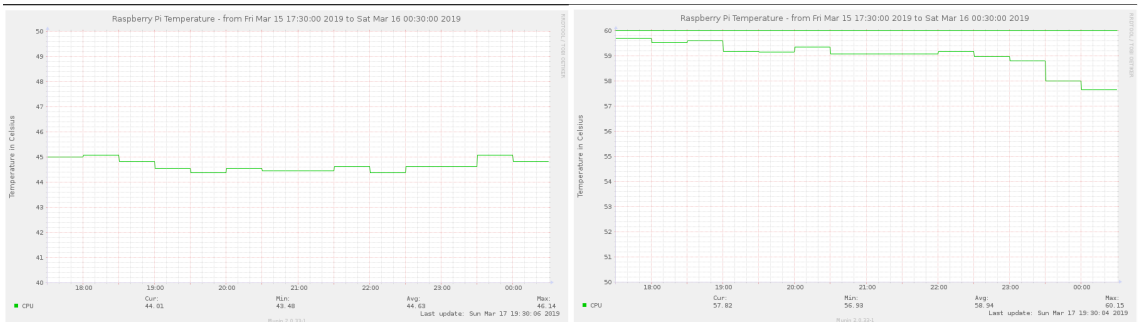

Figure 13.43: Mopidy Client and Server Device CPU Temperature

Volumio

Disk

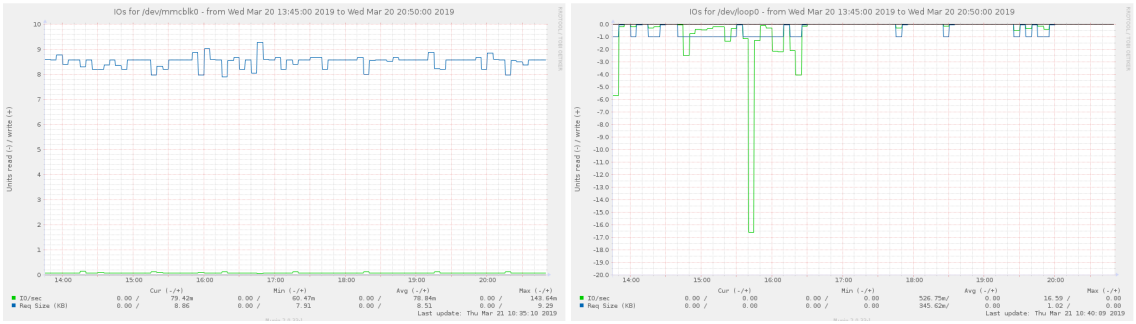

Figure 13.44: Volumio Disk I/O on Client and Server Device

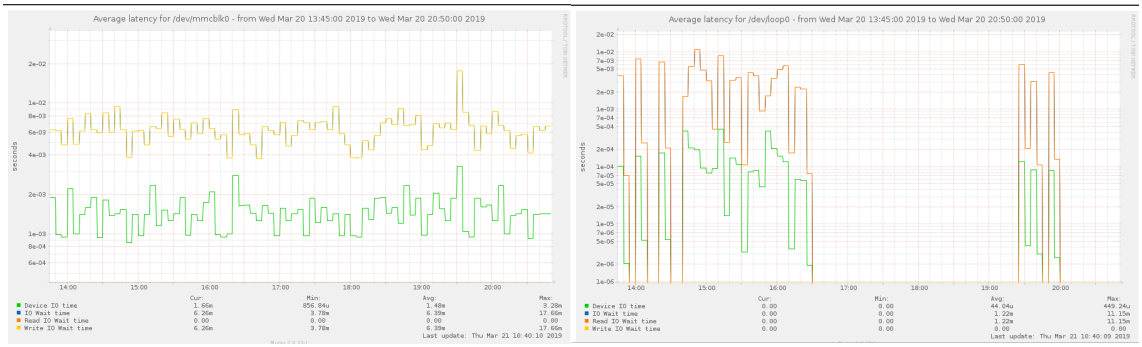

Figure 13.45: Volumio Client and Server Device Disk Latency

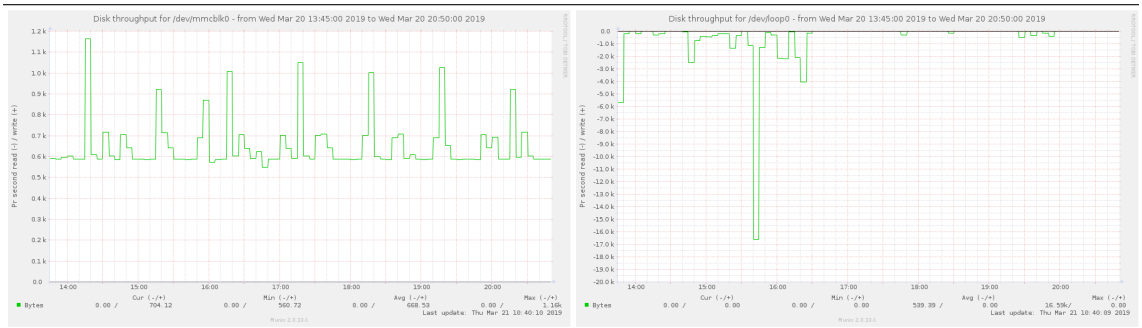

Figure 13.46: Volumio Client and Server Device Disk Throughput

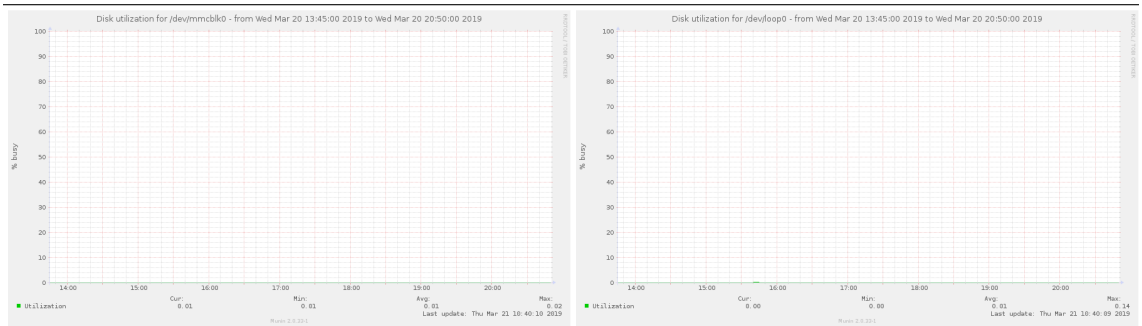

Figure 13.47: Volumio Client and Server Device Disk Utilization

Network

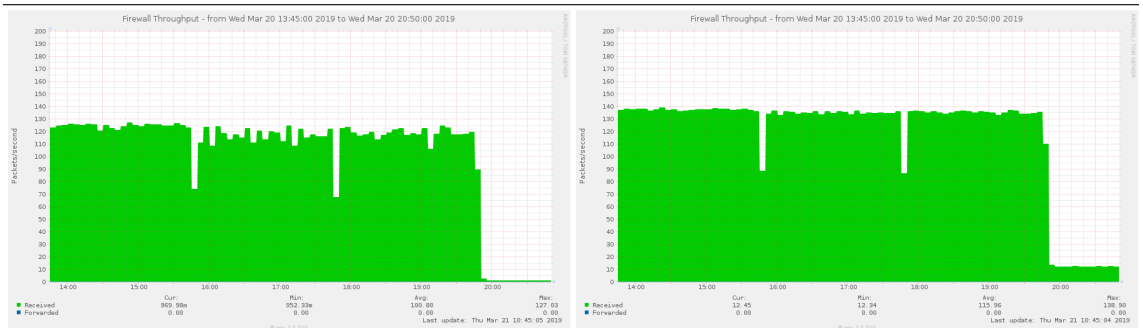

Figure 13.48: Volumio Client and Server Device Firewall Throughput

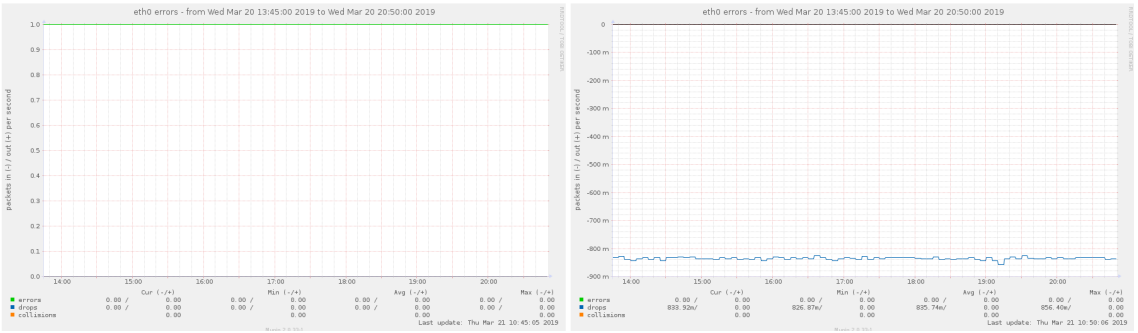

Figure 13.49: Volumio Client and Server Device Eth Errors

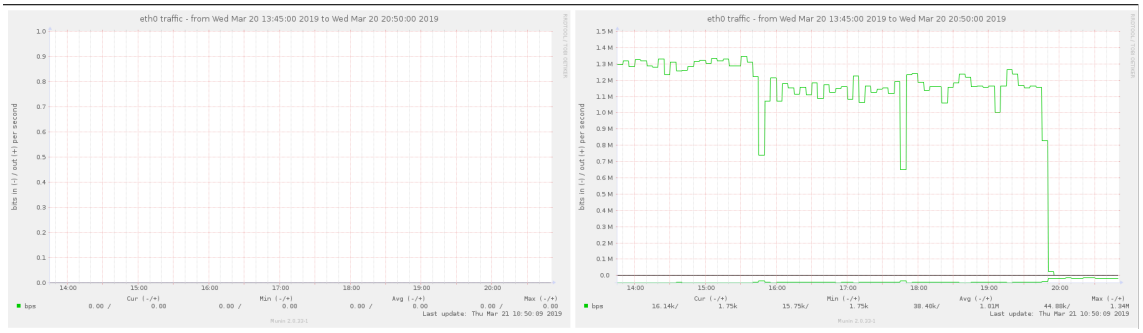

Figure 13.50: Volumio Client and Server Device Eth Traffic

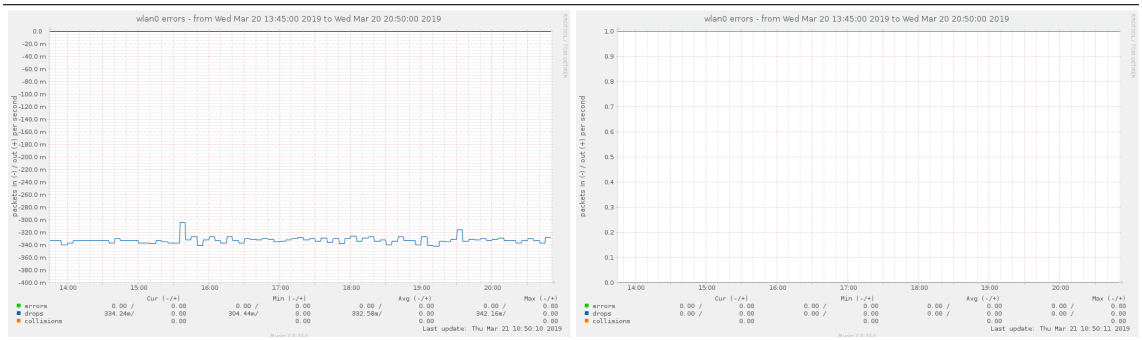

Figure 13.51: Volumio Client and Server Device Wlan Errors

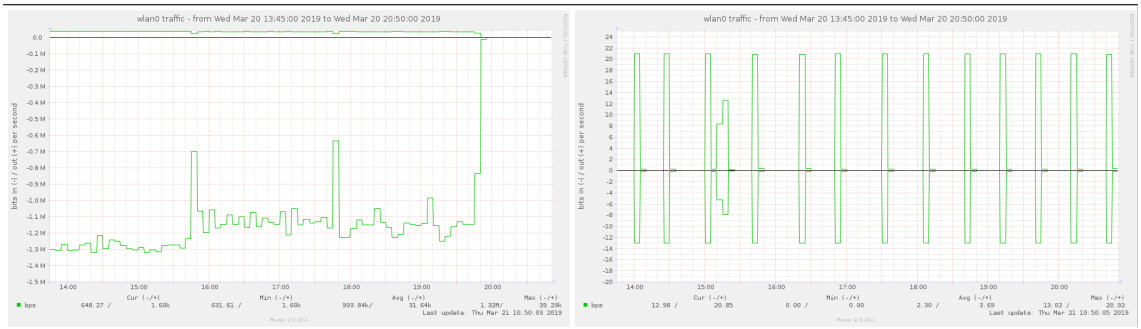

Figure 13.52: Volumio Client and Server Device Wlan Traffic

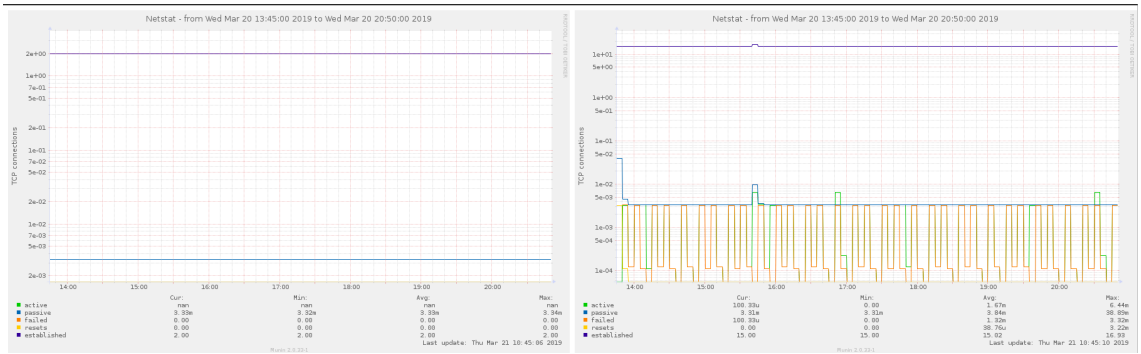

Figure 13.53: Volumio Client and Server Device Netstat

Processes

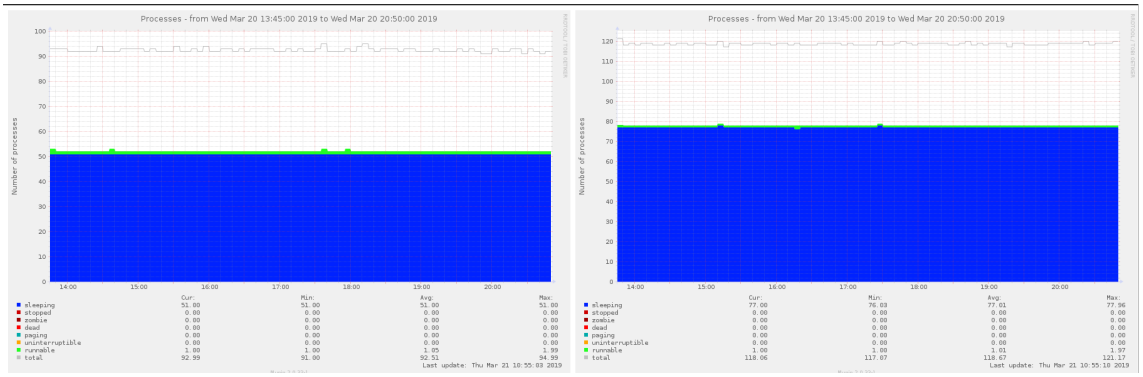

Figure 13.54: Volumio Client and Server Device Processes

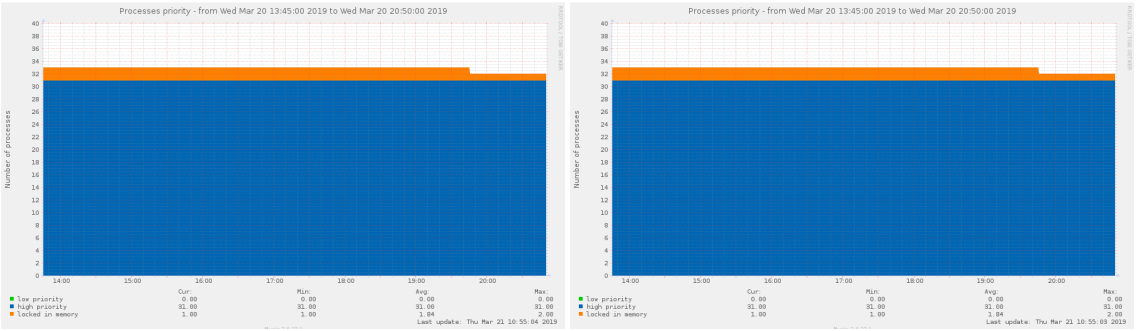

Figure 13.55: Volumio Client and Server Device Processes

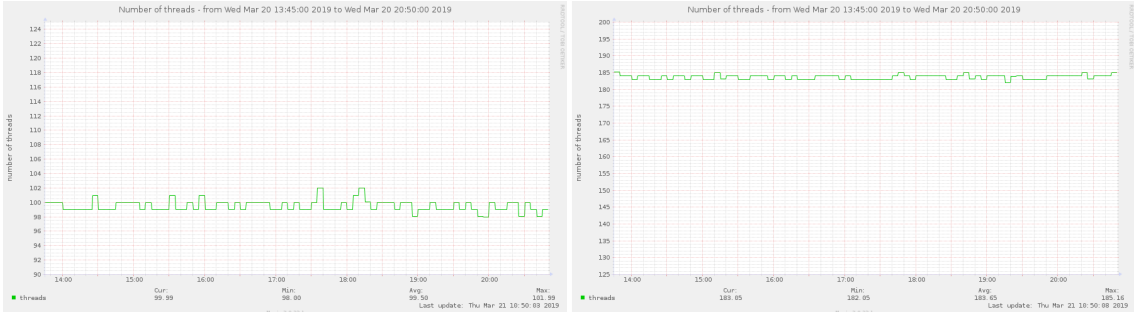

Figure 13.56: Volumio Client and Server Device Number of Threads

System

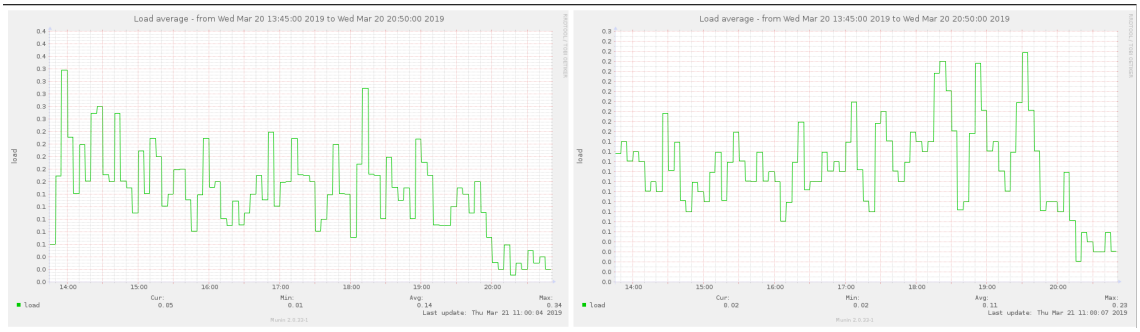

Figure 13.57: Volumio Client and Server Device Load Average

Figure 13.58: Volumio Client and Server Device Individual Interrupts

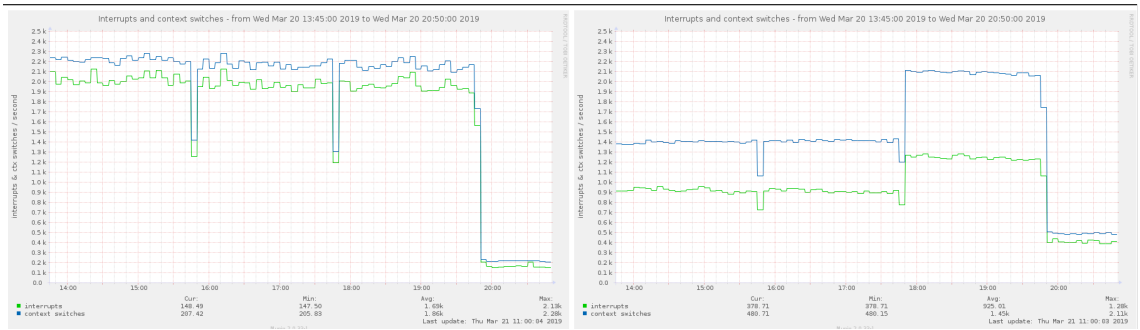

Figure 13.59: Volumio Client and Server Device Interrupts and Context Switches

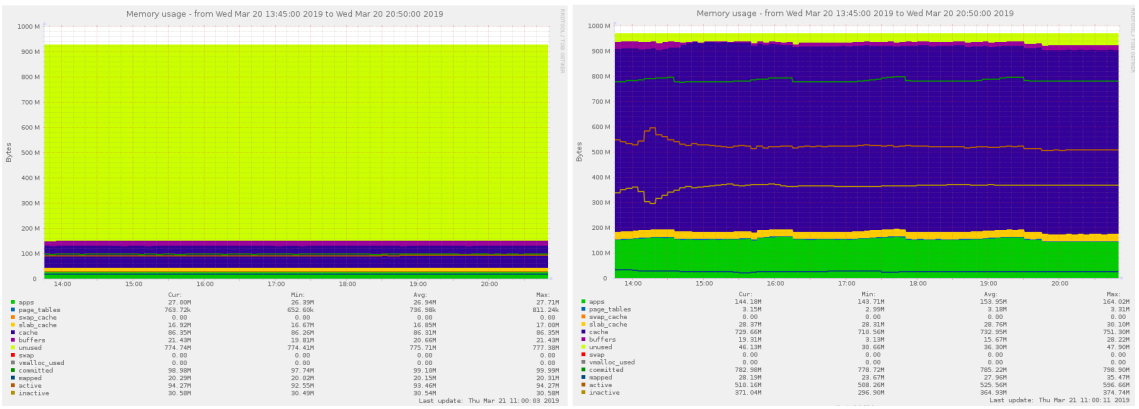

Figure 13.60: Volumio Client and Server Device Memory Usage

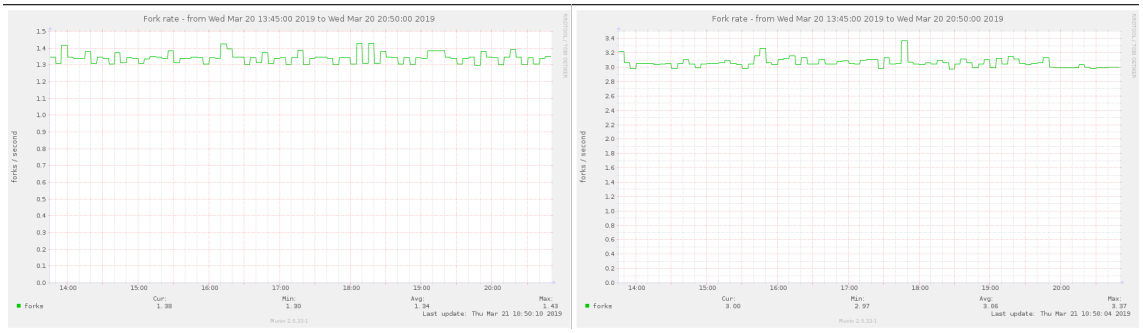

Figure 13.61: Volumio Client and Server Device Fork Rate

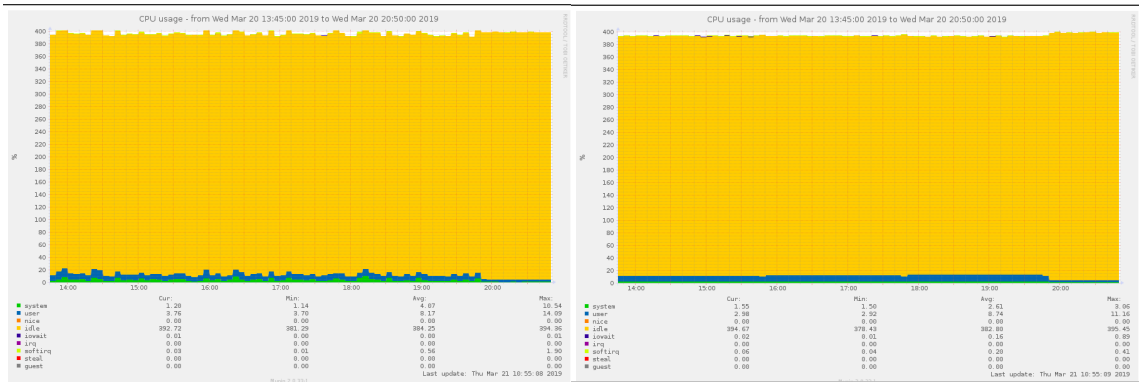

Figure 13.62: Volumio Client and Server Device CPU Usage

Sensors

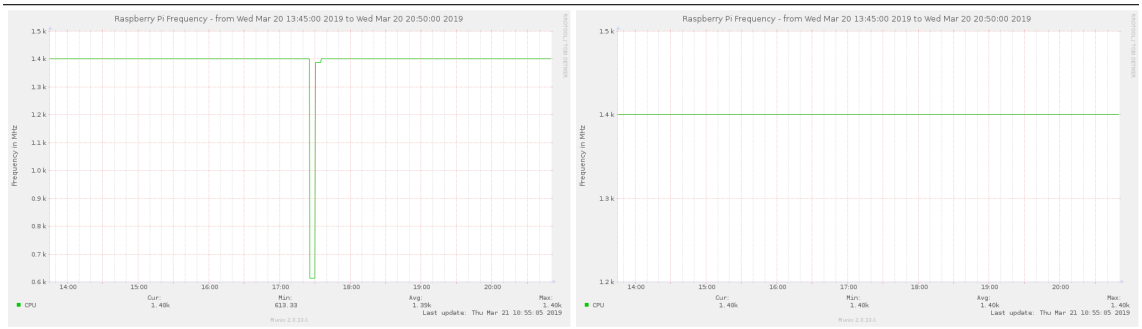

Figure 13.63: Volumio Client and Server Device CPU Frequency

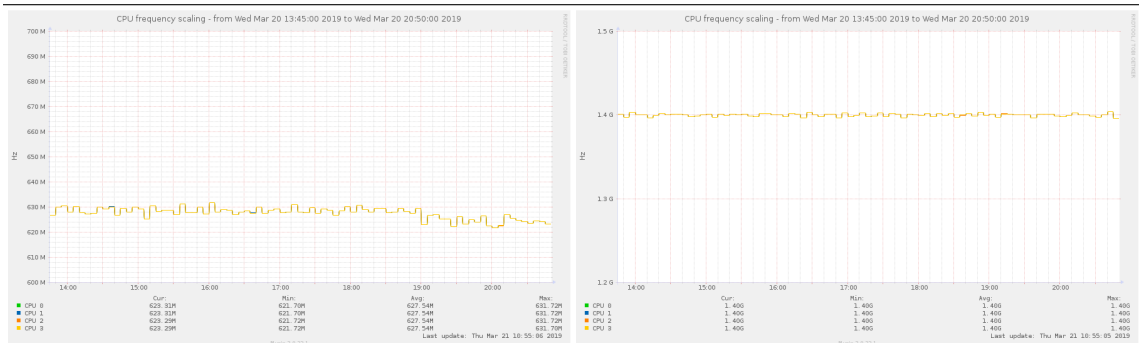

Figure 13.64: Volumio Client and Server Device CPU Frequency Scaling

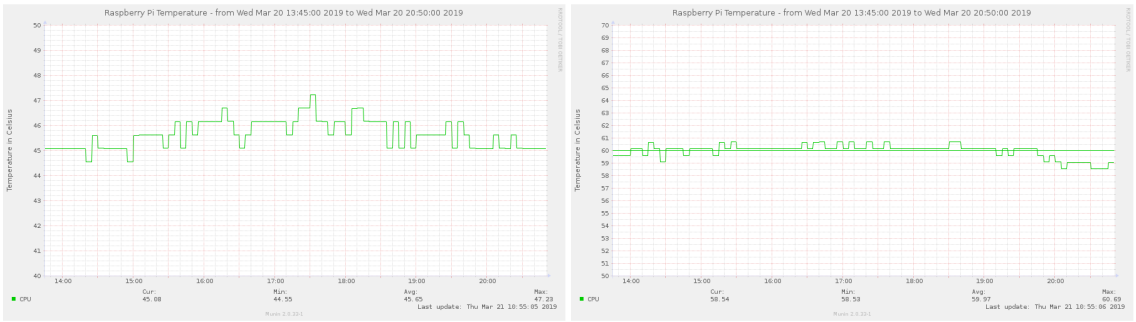

Figure 13.65: Volumio Client and Server Device CPU Temperature

13.2 Audio Server Software Munin Data Tables

MPD

Disk (Client)						
Disk I/O						
	Min		Avg		Max	
	-	+	-	+	-	+
IO/sec	0.00	66.79m	0.00	82.31m	0.00	141.08m
Req Size (KB)	0.00	6.66	0.00	8.46	0.00	10.30
Disk Latency						
	Min		Avg		Max	
Device I/O Time	926.77u		1.60m		2.58m	
I/O Wait Time	4.20		7.32m		16.60m	
Read I/O Time	0.00m		0.00m		0.00m	
Write I/O Time	4.20m		7.32m		16.60m	
Disk Throughput						
	Min		Avg		Max	
	-	+	-	+	-	+
Bytes	0.00	453.23	0.00	697.68	0.00	1.23k
Disk Utilization						
	Min		Avg		Max	
Utilization (%Busy)	0.01		0.01		0.02	

Table 13.1: MPD SnapClient Device Disk Parameters

Disk (Server)						
Disk I/O						
	Min		Avg		Max	
	-	+	-	+	-	+
IO/sec	0.00	69.87m	252.42m	422.49m	1.03	536.13m
Req Size (KB)	0.00	6.40	113.98	6.98	296.23	8.55
Disk Latency						
	Min		Avg		Max	
Device I/O Time	397.44u		43.4m		10.33m	
I/O Wait Time	1.18m		6.21m		15.21m	
Read I/O Time	0.00m		7.65m		20.36m	
Write I/O Time	1.18m		3.00m		9.56m	
Disk Throughput						
	Min		Avg		Max	
	-	+	-	+	-	+
Bytes	0.00	572.00	59.27k	2.82k	185.35k	3.94k
Disk Utilization						
	Min		Avg		Max	
Utilization (%Busy)	0.00		0.39		1.17	

Table 13.2: MPD Server Device Disk Parameters

Network (Client)						
Firewall Throughput (Packets/sec						
	Min		Avg		Max	
	-	+	-	+	-	+
Received	983.67m		99.33		126.69	
Forwarded	0.00		0.00		0.00	
Eth0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	0.00	0.00	0.00	0.00
Wlan0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	270.53m	0.00	302.72m	0.00	343.80m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	669.92	1.58k	939.62k	31.80k	1.26M	39.03k
Netstat (TCP Connections)						
	Min		Avg		Max	
active	nan		nan		nan	
passive	3.31m		3.33m		3.35m	
failed	0.00		0.00		0.00	
resets	0.00		39.22u		3.26m	
established	2.00		2.26		3.00	

Table 13.3: MPD SnapClient Device Network Parameters

Network (Server)						
Firewall Throughput (Packets/sec						
	Min		Avg		Max	
	-	+	-	+	-	+
Received	1.91		99.26		122.94	
Forwarded	0.00		0.00		0.00	
Eth0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	830.03m	0.00	835.49m	0.00	853.19m	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	1.25k	1.31k	23.66k	951.47k	29.07k	1.27M
Wlan0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	327.00m	0.00	335.22m	0.00	349.93m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	186.42	17.12	321.71	22.45	741.32	39.21
Netstat (TCP Connections)						
	Min		Avg		Max	
active	nan		nan		nan	
passive	3.31m		3.33m		3.35m	
failed	0.00		0.00		0.00	
resets	0.00		39.22u		3.22m	
established	4.00		4.26		5.00	

Table 13.4: MPD Server Device Network Parameters

Processes (Client)						
Processes						
	Min		Avg		Max	
	-	+	-	+	-	+
Sleeping	50.00		52.23		56.00	
Uninterruptable	0.00		23.61m		973.33m	
Runnable	1.00		1.05		1.97	
Total	91.03		93.77		99.92	
Number of Threads						
	Min		Avg		Max	
Threads	98.03		100.80		106.97	

Table 13.5: MPD SnapClient Device Process Parameters

Processes (Server)						
Processes						
	Min		Avg		Max	
	-	+	-	+	-	+
Sleeping	52.01		54.27		58.00	
Runnable	1.00		1.04		1.99	
Total	94.01		96.30		100.99	
Number of Threads						
	Min		Avg		Max	
Threads	112.00		114.26		118.98	

Table 13.6: MPD Server Device Process Parameters

System (Client)			
Load Average			
	Min	Avg	Max
Load	0.02	0.13	0.46
Interrupts and Context Switches (/sec)			
	Min	Avg	Max
Interrupts	151.95	1.69k	2.10k
Active	208.25	1.87k	2.27k
Memory Usage (Bytes)			
	Min	Avg	Max
Active	164.04M	165.38M	167.67M
Inactive	51.14M	51.17M	51.21M
Unused	670.26M	673.18M	675.15M
Fork Rate (/sec)			
	Min	Avg	Max
Forks	1.30	1.34	1.50
CPU Usage (%)			
	Min	Avg	Max
System	1.05	3.35	9.27
Idle	381.73	384.41	394.69

Table 13.7: MPD SnapClient Device System Parameters

System (Server)			
Load Average			
	Min	Avg	Max
Load	0.03	0.41	1.03
Interrupts and Context Switches (/sec)			
	Min	Avg	Max
Interrupts	170.32	490.99	550.87
Active	250.84	742.53	898.79
Memory Usage (Bytes)			
	Min	Avg	Max
Active	270.40M	408.59M	437.63M
Inactive	424.17M	541.23M	589.36M
Unused	31.03M	35.88M	43.78M
Fork Rate (/sec)			
	Min	Avg	Max
Forks	1.30	1.39	1.70
CPU Usage (%)			
	Min	Avg	Max
System	1.21	5.89	15.08
Idle	363.71	369.60	394.16

Table 13.8: MPD Server Device System Parameters

Sensors (Client)			
CPU Frequency (MHz)			
	Min	Avg	Max
CPU	600.00	600.00	600.00
CPU Frequency Scaling (MHz)			
	Min	Avg	Max
CPU1	613.87	618.10	620.98
CPU2	613.92	618.10	620.97
CPU3	613.87	610.10	620.99
CPU4	613.92	610.10	620.98
CPU Temperature (°C)			
	Min	Avg	Max
CPU	41.86	42.66	44.00

Table 13.9: MPD SnapClient Device Sensor Parameters

Sensors (Server)			
CPU Frequency (MHz)			
	Min	Avg	Max
CPU	600.00	656.53	1.40k
CPU Frequency Scaling (MHz)			
	Min	Avg	Max
CPU1	624.32	691.30	842.15
CPU2	624.32	691.30	842.12
CPU3	624.30	691.30	842.14
CPU4	624.30	691.30	842.11
CPU Temperature (°C)			
	Min	Avg	Max
CPU	53.69	56.11	58.52

Table 13.10: MPD Server Device Sensor Parameters

Mopidy

Disk (Client)						
Disk I/O						
	Min		Avg		Max	
	-	+	-	+	-	+
IO/sec	0.00	57.86m	0.00	81.74m	0.00	138.22m
Req Size (KB)	0.00	6.63	0.00	8.47	0.00	10.05
Disk Latency (sec)						
	Min		Avg		Max	
Device I/O Time	891.70u		1.68m		3.20m	
I/O Wait Time	4.00		7.65m		12.67m	
Read I/O Time	0.00m		0.00m		0.00m	
Write I/O Time	4.00m		7.65m		12.67m	
Disk Throughput						
	Min		Avg		Max	
	-	+	-	+	-	+
Bytes	0.00	422.65	0.00	695.06	0.00	1.20k
Disk Utilization						
	Min		Avg		Max	
Utilization (%Busy)	0.01		0.01		0.03	

Table 13.11: Mopidy SnapClient Device Disk Parameters

Disk (Server)						
Disk I/O						
	Min		Avg		Max	
	-	+	-	+	-	+
IO/sec	0.00	1.01m	479.63m	2.05	2.39	3.64
Req Size (KB)	0.00	10.33	54.58	11.42	131.06	14.25
Disk Latency						
	Min		Avg		Max	
Device I/O Time	1.62m		3.24m		6.52m	
I/O Wait Time	23.95m		78.49m		105.33m	
Read I/O Time	0.00m		4.62m		10.31m	
Write I/O Time	30.35m		89.96m		185.33m	
Disk Throughput						
	Min		Avg		Max	
	-	+	-	+	-	+
Bytes	0.00	12.98k	54.91k	22.87k	176.23k	45.14k
Disk Utilization						
	Min		Avg		Max	
Utilization (%Busy)	0.24		0.88		2.05	

Table 13.12: Mopidy Server Device Disk Parameters

Network (Client)						
Firewall Throughput (Packets/sec						
	Min		Avg		Max	
	-	+	-	+	-	+
Received	973.78m		96.31		126.42	
Forwarded	0.00		0.00		0.00	
Eth0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	0.00	0.00	0.00	0.00
Wlan0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	313.56m	0.00	330.82m	0.00	344.32m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	181.87	17.17	280.87	22.18	756.88	31.00
Netstat (TCP Connections)						
	Min		Avg		Max	
active	nan		nan		nan	
passive	3.31m		3.33m		3.34m	
failed	0.00		0.00		0.00	
resets	0.00		0.00		0.00	
established	2.00		2.00		2.00	

Table 13.13: Mopidy SnapClient Device Network Parameters

Network (Server)						
Firewall Throughput (Packets/sec						
	Min		Avg		Max	
	-	+	-	+	-	+
Received	3.11		100.40		132.30	
Forwarded	0.00		0.00		0.00	
Eth0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	827.43m	0.00	835.03m	0.00	852.73m	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	1.23k	1.34k	25.65k	961.24k	35.72k	1.34M
Wlan0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	329.11m	0.00	334.61m	0.00	347.44m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	636.24	1.58k	950.23k	34.74k	1.33M	48.40k
Netstat (TCP Connections)						
	Min		Avg		Max	
active	nan		nan		nan	
passive	6.65m		7.26m		26.15m	
failed	0.00		0.00		0.00	
resets	0.00		0.00		0.00	
established	6.00		6.00		6.00	

Table 13.14: Mopidy Server Device Network Parameters

Processes (Client)						
Processes						
	Min		Avg		Max	
	-	+	-	+	-	+
Sleeping	50.03		50.96		51.00	
Uninterruptable	0.00		11.07m		970.00m	
Runnable	1.00		1.06		2.00	
Total	91.00		92.53		94.92	
Number of Threads						
	Min		Avg		Max	
Threads	98.00		99.50		101.90	

Table 13.15: Mopidy SnapClient Device Process Parameters

Processes (Server)						
Processes						
	Min		Avg		Max	
	-	+	-	+	-	+
Sleeping	57.03		58.88		60.97	
Uninterruptable	0.00		0.00		0.00	
Runnable	1.00		1.62		2.98	
Total	100.00		101.47		103.00	
Number of Threads						
	Min		Avg		Max	
Threads	126.00		133.41		137.97	

Table 13.16: Mopidy Server Device Process Parameters

System (Client)			
Load Average			
	Min	Avg	Max
Load	0.02	0.11	0.26
Interrupts and Context Switches (/sec)			
	Min	Avg	Max
Interrupts	151.78	1.65k	2.19k
Active	206.83	1.82k	2.33k
Memory Usage (Bytes)			
	Min	Avg	Max
Active	170.56M	171.03M	171.55M
Inactive	56.72M	56.77M	56.81M
Unused	661.03M	661.75M	662.69M
Fork Rate (/sec)			
	Min	Avg	Max
Forks	1.30	1.34	1.46
CPU Usage (%)			
	Min	Avg	Max
System	1.01	3.43	9.45
Idle	381.26	384.85	394.51

Table 13.17: Mopidy SnapClient Device System Parameters

System (Server)			
Load Average			
	Min	Avg	Max
Load	0.05	0.23	0.40
Interrupts and Context Switches (/sec)			
	Min	Avg	Max
Interrupts	2.22k	2.51k	2.64k
Active	334.59	853.76	1.09k
Memory Usage (Bytes)			
	Min	Avg	Max
Active	161.88M	375.25M	445.90M
Inactive	245.70M	462.17M	701.93M
Unused	26.69M	54.62M	472.22M
Fork Rate (/sec)			
	Min	Avg	Max
Forks	2.38	2.72	2.97
CPU Usage (%)			
	Min	Avg	Max
System	1.68	3.64	7.53
Idle	351.52	365.23	386.77

Table 13.18: Mopidy Server Device System Parameters

Sensors (Client)			
CPU Frequency (MHz)			
	Min	Avg	Max
CPU	600.00	608.89	1.37k
CPU Frequency Scaling (MHz)			
	Min	Avg	Max
CPU1	611.73	616.29	620.02
CPU2	611.73	616.29	620.02
CPU3	611.73	616.29	620.02
CPU4	611.73	616.29	620.02
CPU Temperature (°C)			
	Min	Avg	Max
CPU	43.48	44.63	46.14

Table 13.19: Mopidy SnapClient Device Sensor Parameters

Sensors (Server)			
CPU Frequency (MHz)			
	Min	Avg	Max
CPU	1.40k	1.40k	1.40k
CPU Frequency Scaling (MHz)			
	Min	Avg	Max
CPU1	638.58	646.91	674.50
CPU2	638.58	646.91	674.50
CPU3	638.58	646.91	674.50
CPU4	638.58	646.91	674.50
CPU Temperature (°C)			
	Min	Avg	Max
CPU	56.93	58.94	60.15

Table 13.20: Mopidy Server Device Sensor Parameters

Volumio

Disk (Client)						
Disk I/O						
	Min		Avg		Max	
	-	+	-	+	-	+
IO/sec	0.00	60.47m	0.00	78.84m	0.00	143.64m
Req Size (KB)	0.00	7.91	0.00	8.51	0.00	9.29
Disk Latency						
	Min		Avg		Max	
Device I/O Time	856.84u		1.48m		3.28m	
I/O Wait Time	3.78m		6.39m		17.66m	
Read I/O Time	0.00m		0.00m		0.00m	
Write I/O Time	3.78m		6.39m		17.66m	
Disk Throughput						
	Min		Avg		Max	
	-	+	-	+	-	+
Bytes	0.00	560.72	0.00	668.53	0.00	1.16k
Disk Utilization						
	Min		Avg		Max	
Utilization (%Busy)	0.01		0.01		0.02	

Table 13.21: Volumio SnapClient Device Disk Parameters

Disk (Server)						
Disk I/O						
	Min		Avg		Max	
	-	+	-	+	-	+
IO/sec	0.00	0.00	526.775m	0.00	16.59	0.00
Req Size (KB)	0.00	0.00	345.62	0.00	1.02	0.00
Disk Latency						
	Min		Avg		Max	
Device I/O Time	0.00		44.04u		449.24u	
I/O Wait Time	0.00		1.22m		11.15m	
Read I/O Time	0.00m		1.22m		11.15m	
Write I/O Time	0.00		0.00		0.00	
Disk Throughput						
	Min		Avg		Max	
	-	+	-	+	-	+
Bytes	0.00	0.00	539.39	0.00	16.59k	0.00
Disk Utilization						
	Min		Avg		Max	
Utilization (%Busy)	0.00		0.01		0.14	

Table 13.22: Volumio Server Device Disk Parameters

Network (Client)						
Firewall Throughput (Packets/sec						
	Min		Avg		Max	
	-	+	-	+	-	+
Received	952.33m		100.80		127.03	
Forwarded	0.00		0.00		0.00	
Eth0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	0.00	0.00	0.00	0.00
Wlan0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	304.44m	0.00	332.58m	0.00	342.16m	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	631.61	1.60k	993.84k	31.64k	1.32M	39.29k
Netstat (TCP Connections)						
	Min		Avg		Max	
active	nan		nan		nan	
passive	3.32m		3.33m		3.34m	
failed	0.00		0.00		0.00	
resets	0.00		39.22u		3.26m	
established	2.00		2.00		2.00	

Table 13.23: Volumio SnapClient Device Network Parameters

Network (Server)						
Firewall Throughput (Packets/sec						
	Min		Avg		Max	
	-	+	-	+	-	+
Received	12.34		115.96		138.90	
Forwarded	0.00		0.00		0.00	
Eth0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	826.87m	0.00	835.74m	0.00	856.40m	0.00
Collisions	0.00	0.00	0.00	0.00	0.00	0.00
Eth0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	15.75k	1.75k	38.40k	1.01M	44.88k	1.34M
Wlan0 Errors						
	Min		Avg		Max	
	-	+	-	+	-	+
Errors	0.00	0.00	0.00	0.00	0.00	0.00
Drops	0.00	0.00	0.00	0.00	0.00	0.00
Collisions	0.00		0.00		0.00	
Wlan0 Traffic						
	Min		Avg		Max	
	-	+	-	+	-	+
bps	0.00	0.00	2.30	3.69	13.02	20.92
Netstat (TCP Connections)						
	Min		Avg		Max	
active	0.00		1.67m		6.44m	
passive	3.31m		3.84m		38.89m	
failed	0.00		1.32m		3.32m	
resets	0.00		38.76u		3.22m	
established	15.00		15.02		16.93	

Table 13.24: Volumio Server Device Network Parameters

Processes (Client)						
Processes						
	Min		Avg		Max	
	-	+	-	+	-	+
Sleeping	51.00		51.00		51.00	
Uninterruptable	0.00		0.00		0.00	
Runnable	1.00		1.05		1.99	
Total	91.00		92.51		94.99	
Number of Threads						
	Min		Avg		Max	
Threads	98.00		99.50		101.99	

Table 13.25: Volumio SnapClient Device Process Parameters

Processes (Server)						
Processes						
	Min		Avg		Max	
	-	+	-	+	-	+
Sleeping	76.03		77.01		77.96	
Runnable	1.00		1.01		1.97	
Total	117.07		118.67		121.17	
Number of Threads						
	Min		Avg		Max	
Threads	182.05		183.65		185.16	

Table 13.26: Volumio Server Device Process Parameters

System (Client)			
Load Average			
	Min	Avg	Max
Load	0.01	0.14	0.34
Interrupts and Context Switches (/sec)			
	Min	Avg	Max
Interrupts	147.50	1.69k	2.13k
Active	208.83	1.86k	2.28k
Memory Usage (Bytes)			
	Min	Avg	Max
Active	92.55M	93.46M	94.27M
Inactive	30.49M	30.54M	30.58M
Unused	774.41M	775.71M	777.38M
Fork Rate (/sec)			
	Min	Avg	Max
Forks	1.30	1.34	1.43
CPU Usage (%)			
	Min	Avg	Max
System	1.14	4.07	10.54
Idle	381.29	384.25	394.36

Table 13.27: Volumio SnapClient Device System Parameters

System (Server)			
Load Average			
	Min	Avg	Max
Load	0.02	0.11	0.23
Interrupts and Context Switches (/sec)			
	Min	Avg	Max
Interrupts	378.71	925.01	1.28k
Active	480.15	1.45k	2.11k
Memory Usage (Bytes)			
	Min	Avg	Max
Active	508.26M	525.56M	596.66M
Inactive	296.90M	364.93M	374.74M
Unused	30.66M	36.30M	47.90M
Fork Rate (/sec)			
	Min	Avg	Max
Forks	2.97	3.06	3.37
CPU Usage (%)			
	Min	Avg	Max
System	1.50	2.61	3.06
Idle	378.43	382.80	395.45

Table 13.28: Volumio Server Device System Parameters

Sensors (Client)			
CPU Frequency (MHz)			
	Min	Avg	Max
CPU	613.33	1.39k	1.40k
CPU Frequency Scaling (MHz)			
	Min	Avg	Max
CPU1	621.70	627.54	631.72
CPU2	621.70	627.54	631.72
CPU3	621.72	627.54	631.72
CPU4	621.72	627.54	631.70
CPU Temperature (°C)			
	Min	Avg	Max
CPU	44.55	45.65	47.23

Table 13.29: Volumio SnapClient Device Sensor Parameters

Sensors (Server)			
CPU Frequency (MHz)			
	Min	Avg	Max
CPU	1.40k	1.40k	1.40k
CPU Frequency Scaling (MHz)			
	Min	Avg	Max
CPU1	1.40k	1.40k	1.40k
CPU2	1.40k	1.40k	1.40k
CPU3	1.40k	1.40k	1.40k
CPU4	1.40k	1.40k	1.40k
CPU Temperature (°C)			
	Min	Avg	Max
CPU	58.53	59.97	60.69

Table 13.30: Volumio Server Device Sensor Parameters

Bibliography

[1] BeagleBoard.org, "Beaglebone black," 2018. [Online]. Available: https://beagleboard.org/black

[2] Raspberry Pi Foundation, "Raspberry pi 3 model b+." [Online]. Available: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/

[3] Adafruit, "Overview | adafruit i2s audio bonnet for raspberry pi," 2019. [Online]. Available: https://learn.adafruit.com/adafruit-i2s-audio-bonnet-for-raspberry-pi/overview

[4] The Economist, "Having rescued recorded music, spotify may upend the industry again," 2018. [Online]. Available: https://www.economist.com/business/2018/01/11/having-rescued-recorded-music-spotify-may-upend-the-industry-again?frsc=dg|e

[5] ASUS, "Asus tinker board." [Online]. Available: https://www.asus.com/ie/Single-Board-Computer/Tinker-Board/

[6] HiFiBerry, "Hifiberry dac+ pro," 2019. [Online]. Available: https://www.hifiberry.com/shop/boards/hifiberry-dac-pro/

[7] PulseAudio, "Pulseaudio," 2014. [Online]. Available: https://www.freedesktop.org/wiki/Software/PulseAudio/About/

[8] The Music Player Daemon Project, "Music player daemon 0.21.4 documentation," 2018. [Online]. Available: https://www.musicpd.org/doc/

[9] Volumio, "Volumio documentation," 2018. [Online]. Available: https://volumio.github.io/docs/index.html

[10] J. Park, J. Kim, and S. Kang, "Ble-based accurate indoor location tracking for home and office," Computer Science & Information Technology, 2015.

[11] J. Gjengset, J. Xiong, G. McPhilips, and K. Jamieson, "Phaser: Enabling phased array signal processing on commodity wifi access points," School of Information Systems at Institutional Knowledge at Singapore Management University, 2014.

[12] Y. Heejung and K. Taejoon, "Beamforming transmission in ieee 802.1ac under time-varying channels," The Scientific World Journal, 2014.

[13] A. Uddin Afaz, R. Arablouei, F. de Hoog, B. Kusy, R. Jurdak, and N. Bergmann, "Estimating angle-of-arrival and time-of-flight for multipath components using wifi channel state information," Sensors, 2018.

[14] AltBeacon.org, "Altbeacon - the open proximity beacon," 2019. [Online]. Available: https://altbeacon.org/

[15] Google Developers, "Beacons | google developers," 2019. [Online]. Available: https://developers.google.com/beacons/

[16] Android Developers, "Wi-fi location: ranging with rtt | android developers," 2019. [Online]. Available: https://developer.android.com/guide/topics/connectivity/wifi-rtt

[17] Max, "Creating altbeacon with raspberry pi using bluez example code," 2018. [Online]. Available: https://scribles.net/creating-altbeacon-using-bluez-example-code-on-raspberry-pi/

[18] Radius Networks, "Android beacon library," 2018. [Online]. Available: https://altbeacon.github.io/android-beacon-library/eddystone-how-to.html

[19] A. Reltz, C. Sexton, and F. Varol, "Altbeacon protocol specification v1.0," 2015. [Online]. Available: https://github.com/AltBeacon/spec

[20] J. Pohl, "Snapcast json rpc control api," 2018. [Online]. Available: https://github.com/badaix/snapcast/blob/master/doc/json_rpc_api/v2_0_0.md

[21] Loup, "Beacons | flutter beacons plugin for android and ios," 2018. [Online]. Available: https://github.com/loup-v/beacons

[22] Sky UK, "Sky q hub wireless broadband & broadband router," 2019. [Online]. Available: https://www.sky.com/shop/broadband-talk/sky-hub/

[23] J. Pohl, "Configuring zones," 2016. [Online]. Available: https://github.com/badaix/snapcast/issues/54

[24] N. Bridoux, "Beacon scanner," 2018. [Online]. Available: https://play.google.com/store/apps/details?id=com.bridou_n.beaconscanner

[25] SSH Communications Security Inc., "Ssh (secure shell) home page," 2019. [Online]. Available: https://www.ssh.com/ssh/

[26] Munin, "Munin wiki start," 2019. [Online]. Available: http://munin-monitoring.org/

[27] Admin's Choice, "Crontab - quick reference." [Online]. Available: https://www.adminschoice.com/crontab-quick-reference

[28] Systutorials, "mpc (1) - linux man pages." [Online]. Available: https://www.systutorials.com/docs/linux/man/1-mpc/

[29] BlueZ Project, "Bluez official linux bluetooth protocol stack," 2018. [Online]. Available: http://www.bluez.org/

[30] E. Upton, "Introducing turbo mode: up to 50% more performance for free - raspberry pi." [Online]. Available: https://www.raspberrypi.org/blog/introducing-turbo-mode-up-to-50-more-performance-for-free/

[31] Statista, "Number of spotify premium subscribers worldwide from 1st quarter 2015 to 4th quarter 2018 (in millions)," 2019. [Online]. Available: https://www.statista.com/statistics/244995/number-of-paying-spotify-subscribers/

[32] K. Sehgal, "Spotify and apple music should become record labels so musicians can make a fair living," 2018. [Online]. Available: https://www.cnbc.com/2018/01/26/how-spotify-apple-music-can-pay-musicians-more-commentary.html

[33] W. Page and D. Safir, "Money in, money out: Lessons from cmos in allocating and distributing licensing revenue," Music & Copyright Newsletter, 2018. [Online].

Available: http://serci.org/congress_documents/2018/money_in_money_out.pdf

[34] J. Muikku, "Pro rata anduser centric distribution models: A comparative study," Digital Media Finland, 2017. [Online]. Available: http://www.digitalmedia.fi/wp-content/uploads/2018/02/UC_report_final_171213.pdf

[35] J. Poort, J. Quintais, M. A. van der Ende, A. Yagafarova, and M. Hageraats, "Global online piracy study," SSRN Electronic Journal, 2018. [Online]. Available: https://www.ivir.nl/publicaties/download/Global-Online-Piracy-Study.pdf

[36] MUSO, "2017 annual piracy reports," 2018. [Online]. Available: https://www.muso.com/magazine/global-piracy-increases-throughout-2017-muso-reveals/

[37] Statista, "Global small business streaming without music licenses in 2018," 2019. [Online]. Available: https://www.statista.com/statistics/939333/illegal-music-streaming-by-business-owners-worldwide/

[38] W. Ashford, "Researchers raise privacy concerns about bluetooth low energy devices," 2015. [Online]. Available: https://www.computerweekly.com/news/4500246790/Researchers-raise-privacy-concerns-about-Bluetooth-Low-Energy-devices

[39] N. Dunne, "Proximity marketing: Often creepy, but it doesn't have to be," 2017. [Online]. Available: https://www.digitalethics.org/essays/proximity-marketing-often-creepy-it-doesnt-have-be

[40] A. Barisain-Monrose, "Mpdroid a modern mpd client for android," 2018. [Online]. Available: https://github.com/abarisain/dmix

[41] F. Lutkes, "malp," 2019. [Online]. Available: https://gitlab.com/gateship-one/malp